Fuzzy Sets-based Heuristics

José L. Verdegay

Depto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Granada
18071 Granada, SPAIN

Abstract

The main aim of this paper is to show how fuzzy sets and systems can be used to produce optimization algorithms able of being applied in a variety of practical situations. In this way, and on the one hand, fuzzy sets based heuristics for Linear Programming problems are considered. To show the practical realisations of the proposed approach, the Travelling Salesman Problem is assumed, and a new heuristic proving the efficiency of using fuzzy rules as termination criteria is shown. On the other hand, the basic ideas of a Fuzzy Adaptive Neighborhood Search (FANS) heuristic algorithm are also presented. Its main motivation is to provide a general purpose optimization tool, which is easy to tailor to specific problems by means of appropriate definition of its components. The Knapsack Problem with multiple constraints will serve to show the high solution potential of this another fuzzy sets based heuristic algorithm.

Keywords: Heuristics, Knapsack Problem, Fuzzy Rules, Fuzzy Sets and Systems, Travelling Salesman Problem, Linear Programming, Genetic Algorithms, Simulated Annealing, Decision Support Systems.

1 Introduction

From a broad point of view the interface between two different areas, Decision Support Systems and Fuzzy Sets and Systems, is considered in this paper. On the one hand, the term Decision Support System (DSS) was coined at the beginning of the 70s to feature the computer programs that could support a user in making decisions when facing ill-structured problems. Nowadays, software for supporting decision-making is available for almost any management problem. On the other hand, in the early sixties, based on the fact that classical logic does not reflect, to the extent that it should, the omnipresent imprecision in the real world, L. A. Zadeh proposed the Theory of Fuzzy Sets and Fuzzy Logic. Nowadays Fuzzy Logic is employed with great success in the conception, design, construction and utilisation of a wide range of products and systems whose functioning is directly based on the ways human beings reason.

As it is well known, design, construction and search for exact algorithms solving real life problems are also key objectives in the DSS area and, although this kind of problems generally has a high level of difficulty, they need to be solved because of their importance. Both facts, difficulty and importance, together with increasing computer power, encourage the development of heuristics which, even though it may lead to non-optimal solutions, can solve the problem at hand based on the decision makers' satisfaction. In this way the decision maker may prefer to obtain satisfying solutions according to his (her) wishes than optimal ones.

Consequently, in order to face a problem in terms of satisfaction, and not only optimization, heuristic methods must search for solutions not only providing good values for the objective function, but also showing some additional characteristics. In general, those characteristics will be of a subjective nature and therefore they could be well modelled by fuzzy sets.

The last decades have witnessed a flow of information from classical fields, such as Operational Research or Control Theory, to the area of Fuzzy Sets and Systems, which has provided very fruitful results. However, it has not been so usual to see this continuous interaction in the other direction. In order to bridge this gap, the aim of this contribution is twofold: First one wants to show how fuzzy sets and systems can help to provide robust and adaptive heuristic optimization algorithms able of being applied in a variety of situations. The second objective is to give a view, as broad as possible, on the practical applications that fuzzy sets based heuristics algorithms have by describing two particular fuzzy sets based heuristics.

On the one hand, fuzzzy sets based heuristics for Linear Programming (LP) problems will be considered in the next section. LP provides a very powerful context that has been used in a range of different applications, and has a documented history of success [11, 14]. On the other hand, LP assuming fuzzy parameters, i.e., Fuzzy Linear Programming (FLP) is one of the best studied topics in the field [2, 3, 6]. To show the practical realisations of the approach, the Travelling Salesman Problem (TSP) will be considered, and a heuristic proving the efficiency of using fuzzy rules as termination criteria will shown [5, 13, 15, 16, 17].

In the contextt of DSS, simple general purpose optimization tools are key elements to decision makers, because they enable them to obtain initial solutions with minimal knowledge of the problem to be solved. Such initial solutions may serve as a guide for further improvements. Among those simple approaches are neighborhood or local search based heuristic techniques. Hence, on the other hand, in the last section we will present the basic ideas of a Fuzzy Adaptive Neighborhood Search (FANS) heuristic algorithm. Its main motivation is to provide a general purpose optimization tool, suitable to be embedded in a DSS, which is easy to tailor to specific problems by means of appropriate definition of its components. FANS is termed Fuzzy, because the solutions are qualified in terms of fuzzy valuations, and adaptive, because its behaviour is adapted as a response to the state of the search. Some illustrative results for practical problems will show the high solution potential of this algorithm.

2 Using fuzzy rules for terminating algorithms

The key point in this section is that FLP methodologies may help to find solutions for problems in which to find an optimum solution is not easy. As it is well known there are a lot of NP problems (Knapsack, Travelling Salesman, etc.) which cannot effectively be solved in all cases, but which are of the utmost importance in a variety of real applications. In these problems the decision-maker must usually accept approximate solutions instead of optimum ones. At this point the aim here is to show how the FLP can help classical MP models and techniques by providing approximate (fuzzy) solutions that may be used by the decision-maker as help to quickly obtain a good enough solution for these problems.

Let‘s justify this fact as follows. An algorithm for solving a general classical optimisation problem can be viewed as an iterative process that produces a sequence of points according to a prescribed set of instructions, together with a termination criterion. Usually we are interested in algorithms that generate a sequence $x_1,x_2,...,x_N$ that converges to an overall, optimum solution. But in many cases however, and because of the difficulties in the problem, we may have to be satisfied with less favourable solutions. Then the iterative procedure may stop either 1) if a point belonging to a prefixed set (the solution set) is reached, or 2) if some prefixed condition for satisfaction is verified.

But, the conditions for satisfaction are not to be meant as universal ones. In fact they depend on several factors such as the decision-maker, the features of the problem, the nature of the information available, ... In any case, assuming that a solution set is prefixed, the algorithm will stop if a point in that solution set is reached. Frequently, however, the convergence to a

point in the solution set is not easy because, for example, of the existence of local optimum points, and hence we must redefine some rules for terminating the iterative procedure.

Roughly speaking, the possible criteria to be taken into account for terminating the algorithms are nothing but control rules. Thus these rules could be associated to the two above points: the solution set, and the criteria for terminating the algorithm. As is clear, fuzziness can be introduced in both points, not assuming it as inherent in the problem, but as help for obtaining, in a more effective way, some solution for satisfying the decision-maker's wishes. This is meant so that the decision-maker might be more comfortable when obtaining a solution expressed in terms of satisfaction instead of optimisation, as is the case when fuzzy control rules are applied to the control processes. Therefore, and in the particular case of optimisation problems [5, 17], it makes sense to consider fuzziness

a) In the Solution Set, i.e., there is a membership function giving the degree with which a point belongs to that set, and

b) On the conditions for satisfaction, and hence Fuzzy Control rules on the criteria for terminating the algorithm.

In the particular case of LP problems, if a conventional problem is assumed

$$\text{Min } \{cx / Ax = d;\ x \geq 0\}$$

the Simplex Algorithm, with the usual denotation, can be summarised as follows,

1) Find an initial extreme point x with basis B,

2) Let x be an extreme point with basis B, and let R be the matrix corresponding to the nonbasic variables. Compute $c_B B^{-1} R - c_R$

   If this vector is non positive then stop, x is an optimal extreme point.

   Else select the most positive component $c_B B^{-1} a_j - c_j$ and compute $y_j = B^{-1} a_j$:

   If $y_j = B^{-1} a_j$ is less than or equal to 0 Then stop. Objective unbounded.

   If $y_j = B^{-1} a_j$ is neither less than nor equal to 0 Then go to step 3

3) Find the new extreme point by changing the current basis. Repeat step 1.

Therefore, as may be seen, in the Simplex Algorithm control rules appear mainly in the second step as

- The non positivity of the vector $c_B B^{-1} R - c_R$ could be meant in a soft sense,

- The positivity of the component $c_B B^{-1} a_j - c_j$ could be measured according to some membership function, and

- The accomplishment of $y_j = B^{-1} a_j \leq 0$, if this is viewed as a constraint, could be fuzzified.

If the first possibility is considered, a new second step can be formulated,

2') Let x be an extreme point with basis B. Compute $c_B B^{-1} R - c_R$. If

$\forall j = 1, ..., n,\ c_B y_j - c_j <_f 0,\ c_j \in c_R$

Then stop.

Thus this condition is stated as a fuzzy constraint, meaning that the decision-maker can accept violations in the accomplishment of the control rules, $c_B y_j - c_j < 0$, to obtain a near, and therefore approximate, optimal solution instead of a full optimal one.

In the following, the above fuzzy rules, meant as termination criteria in the algorithms used in

practical realisations of DSS, will be illustrated here by means of a very well known problem: the Travelling Salesman Problem (TSP). TSP finds application in a variety of situations: postal routes, tightening the nuts on some piece of machinery on assembly lines, etc. In short TSP is addressed as follows: Let G be a directed graph in which the nodes represent cities and each edge has assigned a positive cost (the distance between each two cities). If a route of G is defined as a directed cycle that includes every vertex in G, and the cost of a route is the sum of the cost of the edges on the route, the TSP is to find a route of minimum cost.

We denote i = 1 the first city of the route and 2,3,...n the other cities, d_{ij} to distance between the city i and the j one, the value of the variable x_{ij} is 1 if j is the next city in the route to city i and 0 otherwise. If N = {1,2,...,n}, the mathematical formulation of the TSP is:

$$\begin{aligned}
\min \quad & z = \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}\, x_{ij} \\
\text{s.t.} \quad & \sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1,\ldots,n, \\
& \sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1,\ldots,n, \\
& \sum_{i\in Q}\sum_{j\in \overline{Q}} x_{ij} \geq 1 \qquad \forall Q \subset N \\
& x_{ij} \in \{0,1\} \qquad i, j = 1,2,\ldots,n.
\end{aligned}$$

In order to introduce fuzzy termination criteria in the exact algorithms of the TSP, we consider that the value of a TSP optimal solution is not a crisp unknown value, but a vague value, because in a great dimension TSP, for which the exact algorithms known need a lot of time to obtain an optimal solution, the decision maker can be comfortable in having an almost optimal solution instead of the very optimal one. In such a situation the optimal value can be seen as a fuzzy set on $[L_0,U_0]$ defined by a membership function as:

$$m(z) = \begin{cases} 1 & si \quad z < L_0 \\ f(z) & si \quad L_0 \leq z \leq U_0 \\ 0 & si \quad z > U_0 \end{cases}$$

where $f(z) \in [0,1]$ $\forall z \in [L_0,U_0]$, is a not increasing continuous function, L_0 a lower bound and U_0 a upper bound of the optimal value of the TSP which shall be determined "a priori" as it will be shown. As usual, this membership function shows that if the value z of a TSP route is greater than U_0 then is not allowed by the decision maker. A lower value to L_0 can be a good solution and values between L_0 and U_0 are admissible, but the level of admission will be increasing when z decreases. Obviously, the highest level of admission is obtained when z is equal to L_0.

If the decision maker accept a not optimal solution with a membership degree not lower than α ($0<\alpha<1$), a termination criterion is:

$$\mu(z) \geq \alpha \qquad \text{or} \qquad z \leq f^{-1}(\alpha) \tag{1}$$

The values of L_0, U_0 and the function f must be the correct ones in order to provide the expected results by the decision maker. Unsuitable values of L_0, U_0 can produce solutions with great errors. Equally, an incorrect function f can cancel out the flexibility. From [13, 17] it follows that in such situations a good option is to use a concave function in the definition of the membership function, concretely a function as

$$f(z) = \sqrt[n]{\frac{U_0 - z}{U_0 - L_0}} \tag{2}$$

for which the bounds L_0 and U_0 can be computed by suitable efficient existing algorithms.

In order to illustrate the use of fuzzy termination criteria in the TSP, the well known algorithm by Little et al. [7], that here is denoted LMSK algorithm in short and that is specially designed for solving TSP, has been considered.

2.1 LMSK algorithm.

This is a branch and bound algorithm that uses relaxation of TSP as a matching problem denoted by PA (TSP). The algorithm starts by solving the PA (TSP) by the Hungarian Method; if the obtained solution not possess sub-routes then it is an optimal solution of the TSP. Else the algorithm proceeds to branch. Each iteration, one chooses the most recent sub-problem TSP_k among the unsolved ones. If the optimal value is lower than the best current value, then it is saved as the best current value, or alternatively one branches according to this problem has sub-routes or do not. If the optimal value is equal to or greater than the best current value, then one rejects the sub-problem and start another iteration. The rule for branching consists in choosing a variable x_{ij} and to obtain two sub-problems by assigning 0 and 1 values to the selected variable. The process terminates when do not exists any unsolved sub-problem.

In a TSP_k (node k) sub-problem, as a consequence of the above branching, there are variables x_{ij} with fixed values (0 or 1). In graph terms, there are an (i,j) edge included or not in the route. We denote I the included edge set and E the excluded edge set. Then, TSP_k can be describe as:

$$\begin{aligned} \min \quad & \sum_{(i,j)\in I} d_{ij} + \sum_{i\in S}\sum_{j\in T} d'_{ij}\, x_{ij} \\ \text{s.a.} \quad & \sum_{i\in S} x_{ij} = 1, \quad \forall j \in T, \\ & \sum_{j\in T} x_{ij} = 1, \quad \forall i \in S, \\ & x_{ij} \in \{0,1\}, \quad \forall (i,j), \quad i \in S, \quad j \in T \end{aligned}$$

where:

$$S = \{i/(i,j) \notin I \ \ \forall j\}, \quad T = \{j/(i,j) \notin I \ \ \forall i\} \quad \text{y} \quad d'_{ij} = \begin{cases} d_{ij} & (i,j) \notin E \\ \infty & (i,j) \in E \end{cases}$$

furthermore d_{ij} are the coefficients of the matrix of reduced distance of previous node, that is to say, the matrix that rest after to obtain the optimum assignation in the previous node.

This sub-problem is solved by using the Hungarian Method. If the obtained solution has sub-routes one proceeds to branch. A rule for branching is to choose a variable x_{rs} where $r \in S$ and $s \in T$, and to make two nodes assigning value 1 or 0 each one. Little et al [7] suggest to select a variable x_{rs} with value 0 if this variable has the maximal potential of increasing in the objective function of the sub-problem. In order to make it, let

$$\{\bar{d}_{ij}\} \quad i \in S, \quad j \in T$$

be the reduced cost of the optimum solution of the sub-problem. Then, for each edge (i,j), $i \in S$, $j \in T$ with reduced cost 0, we compute:

$$p_{ij} = min\{\bar{d}_{ik} \ / h \in T - \{j\}\} + min\{\bar{d}_{hj} \ / h \in S - \{i\}\}$$

which is the minimum amount to increase the optimum value of the assignation to the subproblem, if the chosen variable is fixed to 0. Therefore we can choose x_{rs} such that:

$$p_{rs} = max\{p_{ij} \ / i \in S, \ j \in T, \ \bar{d}_{ij} = 0\} \tag{3}$$

when the variable of branching x_{rs} is chosen, all the new nodes can be obtained making x_{rs} =1 and x_{rs} =0. In the first new node, I has the edge (r,s) as new element, and in the second new node, E has the edge (r,s) as new element.

2.2 Steps of LMSK algorithm

Step 1: [Starting] Let U = ∞ (best bound and real value) and L = {TSP} (subproblem list).

Step 2: [Selecting a sub-problem] If L = Φ then one terminates the process, because the route associated to U is an optimal one (if U= ∞ , the TSP has not solution).

If L ≠ Φ, one chooses the more recent sub-problem TSP_i , and one removes it from the list L. Go to step 3.

Step 3: [Upper bound determination] Solve PA(TSP_i) by means of the Hungarian Method. Let Z_i be the obtained value.

If $Z_i \geq U$, go to step 2.

If $Z_i < U$ and the solution is a route for TSP (there are not subroutes) then make $U = Z_i$.

If $Z_i < U$ and the solution is not a route for TSP (there are subroutes) go to step 4.

Step 4: [Branching] Choose x_{rs} according to (3) and generate two news sub-problems TSP_{i1} and TSP_{i2} by fixing xrs = 0 and xrs = 1. Let $L = L \cup \{TSP_{i1}, TSP_{i2}\}$.

Go to step 2.

Remark: Note that the termination criterion of this algorithm is L ≠ Φ.

2.3 Fuzzy termination criteria in the LMSK algorithm

To introduce a fuzzy termination criterion in the LMSK algorithm, we make a change at the starting step in order to determinate the bounds L_0 and U_0 . L_0 is computed by using the method proposed in [12], and the upper bound U_0 is computed by means of the process described in [10]. In the same starting step, the decision maker will choose and fix α (the lowest level of admission). Finally, at step 2 one must include the fuzzy termination condition (1). Therefore the following new algorithm is obtained:

Step 1: [Starting] Let U = ∞ (best bound and real value) and L={TSP} (subproblem list). Solve by means of the Hungarian Method PA(TSP). If optimum matching is a route of TSP go to step 2. Else, go to 1'.

Step 1': Find L_0 and U_0, then make $U = U_0$ (best real bound) and go to step1'';

Step 1'': Fix α (0<α≤1). If 0<α<1 let $z_0 = f^{-1}(\alpha)$ (bound for the admissible solution, where f is as in (2)). If L ≠ Φ go to step 2. Else, go to step 4.

If α = 1 (the decision maker do not want to improve an admissible solution), let L = Φ and go to step 2.

Step 2: [Selecting a sub-problem] If L = Φ or $U \leq z_0$ stop the process, as the associated route with U is admissible; if L ≠ Φ go to step 1''. Otherwise stop.

If L ≠ Φ and $U > z_0$, select the more recent problem TSP_i , remove it from the list L and go to step 3.

Step 3: [Upper bound determination] Solve PA(TSP_i) by means of the Hungarian Method. Let Z_i be the obtained value.

If $Z_i \geq U$, go to step 2.

If Z_i <U and the solution is a route for TSP (there are not subroutes) then let U = Z_i.

If Z_i <U and the solution is not a route for TSP (there are subroutes) go to step 4.

Step 4: [Branching] Choose x_{rs} according to (3) and generate two news sub-problems TSP_{i1} and TSP_{i2} by fixing x_{rs}=0 and x_{rs}=1. Take L = L$\cup${TSP_{i1},TSP_{i2}}.

Go to step 2.

The introduction of the fuzzy termination criterion on the algorithm has made it more flexible. Now, the decision maker can control the iterations because at step 1'' he can introduce little values for α and to increase them if he want to improve the admissible solution. Consequently the decision maker will take into account the time used for obtaining admissible solutions.

For the sake of illustration, let consider finally the following TSP of 10 cities, with a distance matrix given by:

$$(d_{ij}) = \begin{bmatrix} - & 1 & 62 & 56 & 54 & 27 & 30 & 27 & 55 & 60 \\ 90 & - & 66 & 77 & 52 & 98 & 12 & 55 & 7 & 64 \\ 30 & 41 & - & 60 & 59 & 17 & 72 & 82 & 76 & 21 \\ 57 & 33 & 33 & - & 64 & 78 & 62 & 24 & 70 & 72 \\ 95 & 32 & 69 & 74 & - & 97 & 94 & 92 & 96 & 55 \\ 29 & 25 & 40 & 61 & 25 & - & 27 & 81 & 57 & 94 \\ 98 & 52 & 8 & 11 & 89 & 61 & - & 55 & 91 & 37 \\ 52 & 50 & 90 & 33 & 64 & 86 & 37 & - & 91 & 88 \\ 45 & 83 & 31 & 79 & 70 & 22 & 46 & 18 & - & 91 \\ 96 & 62 & 88 & 9 & 2 & 67 & 64 & 43 & 85 & - \end{bmatrix}$$

We consider the diagonal elements of the matrix and the distances of excluded edges in the iterations with a value M =10 x (max d_{ij}). Then, d_{ii} = M and d_{ij} = M if the edge (i,j) is excluded of a possible route. Then solving the problem with the exact algorithm LMSK, one obtains the optimal route "1→ 2→ 9→ 6→ 5→ 10→ 4 →8→ 7→ 3→ 1" with a total distance z = 218, after solving 15 sub-problems (original problem included).

On the other hand when using the LMSK algorithm with a fuzzy termination criterion, a function f as (2), n = 2, and bounds L_0 = 208 and U_0 = 308 (at the starting step 1'), the admissible solutions for the different values of α is shown in the following table:

α	$z_0=f^{-1}(\alpha)$	Admissible route	Admissible value	Sub-problems solved (It.)
0.5	283	1-7-3-10-5-2-9-8-4-6-1	258	8
0.8	244	1-8-7-4-3-19-5-2-9-6-1	221	10
0.94	219.64	1-2-9-6-5-10-4-8-7-3-1	218	14

Table: Admissible solutions for the example (LMSK algorithm and fuzzy termination criteria)

One can observe that for α = 0.8, an admissible solution is obtained by solving only the 66% of all the sub-problems than the classical algorithm solves. However, the admissible value obtained is very close to the optimum. It is then evident that the saving in time is upper in comparison with the difference between the admissible value and the optimum value. Furthermore for α = 0.94 one obtains an admissible solution which is the exact one, and by performing less iterations than the original classical algorithm. The more the number of cities are in the TSP, the more these advantages are evident.

3 FANS: A Fuzzy Adaptive Neighborhood Search algorithm

Neighborhood based search algorithms, as Simulated Annealing, Tabu Search or Evolutionary Programming, has been applied with success on combinatorial and numerical optimization problems. Here we present the basic ideas of a Fuzzy Adaptive Neighborhood Search (FANS) algorithm, whose main motivation is to provide an optimization tool where experts' knowledge or "rules of thumb"' could be expressed and tested using fuzzy concepts and heuristic decision rules, leading to an expressive and powerfull system. By means of a fuzzy valuation, represented in FANS by fuzzy sets, a fuzzy measure of the generated solutions is obtained. Fuzzy valuations may represent concepts like "Acceptability", "Goodness", etc. Thus a degree of acceptability (goodness) is calculated for each solution and such degrees are used by FANS at the decision stages. The use of schedulers enable FANS to modify its behaviour as a function of the search state. In particular, the operator used to generate solutions is changed when the search seems trapped, leading to an intermediate escaping mechanism. When this mechanism fails, a classical restart operator is applied.

Being a heuristic, FANS needs some parameters and components to be defined. Suitable component definitions and parameters, lead FANS to reflect the behaviour of traditional techniques like Hill Climbing, Random Walks, etc. In [8, 9], we showed how FANS outperformed a Genetic Algorithm (GA) and Simulated Annealing (SA) over a set of real function minimization problems when the three algorithms were given a fixed amount of cost function evaluations. In those problems the only restriction was the range available for each variable, so it was easy to ensure that all the generated solutions were feasible. Here, we will test FANS over Knapsack Problems (KP) with multiple restrictions in order to confirm its potential as a general purpose optimization technique. In this case, infeasible solutions will exist and the algorithm must deal with them in some way.

3.1 FANS components

FANS operation relies on four main components: an operator, to construct new solutions; a fuzzy valuation, to qualify them; an operator scheduler, to adapt the operator's behaviour; and a neighborhood scheduler, to generate and select a new solution.

Modification operator: We must provide an operator OP to produce new solutions s* from the current one s. This operator has to ensure that all the points of the search space could be reached and must be suitable of adaptation in some sense, for example providing tunnable parameters to control its operation.

Each application of OP(s) must return a different solution s_i, meaning that some randomization element must be present in the operator definition. When binary solutions are considered, a typical modification operator is the bit-Flipping one, which randomly changes k values (1's by 0' and vice versa). Higher values of k means very disruptive moves, while lower values leads to less disruptive ones. Clearly k is a suitable parameter to be adjusted. If real valued solutions are considered, any operator that changes the variables in some amount p within the available range could be used. In this case p is the adjustable parameter.

Fuzzy Concepts: Recall that repeated applications of the operator OP enable us to obtain a set of solutions F(s) from a given one s. The fuzzy concept must provide some kind of measure for the solutions $s_i \in F(s)$ with respect to s. So, given the current solution s we could think of how similar are s and s_i , or how close they are, or even how different is s_i from s. In this way, we will be able to provide some classification of the generated solutions. The fuzzy concepts Similar, Close, Different, ... would be represented by appropriated membership functions μ and FANS will deal with "degrees of membership".

Operator Scheduler: This procedure encapsulates the strategy used for the adaptation of the modification operator OP. Both elements are tightly coupled. The scheduler will be executed when certain conditions are met. For example, when the search is trapped in a local minimum, or no progress has been done in certain number of iterations. The action to take will be the adaptation of those tunnable parameters of the operator, and the type of adaptation could be fixed or decided in terms of some running statistics like the current iteration number, the total iterations available, the improvement ratio, or anything else. Whatever the election, the scheduler must return a modified (in some sense) OP operator.

Neighborhood Scheduler: This procedure, using the current operator OP, the membership function of the defined fuzzy concept μ and the current solution s, must return a neighborhood solution. The neighborhood of a solution s is defined as follows:

$$N(s) = \{s^* | \mu(s^*) > 0\}$$

where the solutions s* are obtained by repeated applications of OP(s). In other words, N(s) is the set of solutions which verifies the used concept in any degree. In order to achieve this goal, the scheduler contains 2 procedures: the generator, and the selector. As their names implies, the first one, given the current solution s, the current operator OP and the membership function μ, has to obtain solutions from N(s) sampling the search space with OP. The second procedure must determine which $q \in N(s)$ would be returned; for example “the best one”, or “anyone", or more complex rules as “if the number of iterations done is high, return the best one”. Several situations could arise in the scheduler, for example when the generator could not obtain any solution $s^* \in N(s)$. In such a case, the interaction between components must ensure that any of the elements used by the generator changes. In this way, the generator would change its behaviour leading to a possibly different result the next time it's executed.

In order to apply FANS to a particular problem, these components must be defined. As usual, when more problem-dependant definitions are used, the better the performance. We decide to use simple definitions in order to test the quality of the search strategy induced by FANS. Below, we describe the definition proposed for each component of FANS in order to to deal with the KP with multiple constraints

Modification Operator k-BitFlip: randomly chooses k positions and flips the associated bit value. “Back mutation” is not allowed.

Fuzzy Valuation (Acceptable): the generated solutions will be qualified in terms of “acceptability”, a concept reflecting the following idea: with a solution at hand, those generated solutions improving the current cost, will have a higher degree of acceptability than those with lower cost. Solutions diminishing the cost a little, will be considered “acceptable” but with lower degree. Those solutions demeliorating to much the current cost will not be considered as acceptable. So, given an objective function f , the current solution s, q a neighbor solution, and β the limit for what is considered as acceptable, the following definition comprise those ideas:

$$\mu(q,s,\beta) = \begin{cases} 0 & \text{if } f(q) < \beta \\ (f(q) - \beta)/(f(s) - \beta) & \text{if } \beta \le f(q) \le f(s) \\ 1 & \text{if } f(q) > f(s) \end{cases}$$

Here we will use $\beta = f(s)\cdot(1-\gamma)$, $\gamma \in [0,1]$, and will set $\gamma = 0.05$

Operator Scheduler: the k-BitFlip operator will be adapted trough changes on the k parameter. The used scheme is rather simple: being k_t the actual value, then k_{t+1} will be a random integer value in $[1; 2\ k_t]$. Also, if $k_{t+1} > top = n/10$, then $k_{t+1} = top$.

Neighborhood Scheduler: given the cost function f , the operator OP, the fuzzy valuation and the current solution s, we define the operational neighborhood of s as

$$N(s) = \{x^* | x^* = OP(s)\}$$

and the "semantic" neighborhood of s as

$$N^*(s) = \{x^* | \mu(f(x^*)) > \lambda; x^* \in N(s)\}$$

Taking into account both definitions, we provide two neigborhood schedulers.

Quality Based Grouping Scheme R|S|T: it tries to generate R "Acceptable" neighborhood solutions in maxTrials trials, then those solutions are grouped into S sets on the basis of their acceptability degree, and finally T solutions are returned [9]. We use a 5|3|1 scheme with maxTrials = 12. The S = 3 sets or clusters are represented by overlapped triangular membership functions with boundaries adjusted to fit the range [λ; 1:0], being λ = 0.99 the minimun level of acceptability required. The sets represents the terms Low, Medium, High for the linguistic variable "Quality". At the end of the process, T = 1 solution must be returned. Here, we choose to return any solution of the highest quality available.

First Found Scheme: at most maxT rials = 12 trials are available to obtain a solution x* in N*(s). The first one found is returned.

Fig. 1 shows FANS pseudo code. The iterations end when some external condition holds. The neigborhood scheduler NS is called at the beginning of each iteration, with parameters: current solution S_cur ; fuzzy valuation μ() and modification operator OP. Two situations may ocurr: an "acceptable" neigborhood solution S_new was found or not. In the first case S_new is taken as the current solution and μ() parameters are adapted. In this way, we are varying our notion of "Acceptability" as a function of the context.

If NS could not return any acceptable solution, an exception condition (OK = false) is raised. No solutions were acceptable in the neighborhood induced by the operator. In this case, the operator scheduler OS is executed, returning a modified version of OP. The next time NS will have a modified operator to search for solutions. The trappedCondition() exception is raised when Top iterations were done without improvements in the best solution found. In this case, the doRestart() procedure is executed applying a perturbation operation over the current solution: 1/3 of randomly chosen variables in one are set to zero. Then, the cost of the current solution is reevaluated and μ() is adapted. Then the process is restarted.

```
Procedure FANS
Begin
    While ( not-finalization ) Do
        NS->Run(OP;μ();Scur;Snew; ok);
        If (ok) Then
            Scur = Snew;
            adaptFuzzyVal(μ();Scur;
        Else
            OS->Run(Oper);
        EndIf
        If (trappedCondition()) Then
            doRestart();
        EndIf
    EndWhile
End.
```

Figure 1: FANS Pseudocode

3.2 Algorithms solving the Knapsack Problem with multiple constraints

Among all the large variety of possible KP which can be addressed, we will focus here on the KP with multiple constraints, which can be mathematically formulated as follows

$$\text{Max } \{\sum_{i=1..n} p_i \cdot x_i \,/\, \sum_{i=1..n} w_{ij} \cdot x_i \leq c_j;\ j = 1...m\}$$

Where n is the number of items, m is the number of restrictions, $x_i \in \{0,1\}$ indicates if the i-th item is included or not in the knapsack, p_i is the profit associated with the i-th item, and finally $w_{ij} \in [0,..,r]$ is the weight of i-th item with respect to j-th constraint ($\sum_{i=1..n} w_{ij} > C_j$, $w_{ij} < C\ j$).

For our experiments we select 9 instances from a standard set of 55 problems available from [1]. The instances are named pb5, pb7, Weing1, Weing3, Weing7, Weish10, Weish14, Weish18, Weish27. The number of variables range from 20 to 105 with 2 to 30 restrictions.

To solve these problems and test the performance of the heuristic presented, we compare FANS with two other heuristics: A Simulated Annealing (SA) algorithm and a Genetic Algorithm (GA). Knapsack solutions are represented by binary vectors X, where position i represents the variable x_i. This representation is used in FANS, SA and GA. In our experiments, infeasible solutions will not be taken into account and they will be discarded. No reparation procedure will be used. Initial solutions for each method contains just a unique position in 1.

The GA used may be regarded as "traditional". Mutation is applied to all individuals with certain probability. As mutation operator, the k-BitFlip operator is used with k = 2. If the solution obtained after mutation is infeasible, it is discarded and at most four more trials are done to obtain a feasible one. If no feasible solution was obtained, the original one is kept.

As crossover operator, two classical ones are implemented: 1 point and uniform crossover. In this way we obtain 2 algorithms: GAOP y GAUX respectively. Elitism is also used in both versions. Other parameters are population size PopSize = 100; crossover and mutation probabilities P(xover) = 0:8; P(mut) = 0:2, and Tournament Selection with tournament size q = 2 within a ($\mu = 50 + \lambda = 75$) scheme.

Our implementation of SA is simple and follows the guidelines presented in [4]. The k-BitFlip operator is also used with k = 2. The initial temperature was $T_0 = 5$ and proportional cooling is used with $T_{k+1} = T_k \cdot \alpha$ with $\alpha = 0.9$. Temperature is adapted when 15 neighbors were accepted, or when $2 \cdot n$ neighbors were generated, being n the dimension of the problem. The values for the parameters were empirically determined after a reduced set of experiments.

Now, in order to analyze the performance of FANS, we conduct a set of experiments and made comparisons among the following five algorithms: FRST, is FANS with scheduler R|S|T ; FFF , is FANS with scheduler FirstFound; GAUX, is the GA with uniform crossover; GAOP, is the GA with one-point crossover; and SA, the Simulated Annealing implementation. We want to compare the performances of the algorithms under none or minimal knowledge of the problem (reflected by the use of very simple operators), and when they are given a fixed number of resources (i.e cost function evaluations and number of generated solutions). Thus we can assume the results are consequence of the search strategy and not of additional knowledge.

For each problem and algorithm 30 runs were made; each one ending when maxEvals = 15000 cost function evaluations were done or when maxEvals $\cdot$ 4 solutions were generated. This limit is needed because only feasible solutions are evaluated. The results are analyzed in terms of the error for each problem and globally over the whole test set. The first results are presented on Table 1, where part (a) shows the mean of the errors over 30 runs for each algorithm on each problem. The error is calculated as:

error = 100* ((Optimum - Obtained Value)/(Optimum))

The Table shows that both versions of FANS achieved the lower values, except for problems Weing3, Weing7 y Weish27. \Sa achieved the better value on Weing7 and GAux did it on the other two problems. On part (b) of Table 1, an × indicate if the algorithm on the column reached the optimum of the problem on the row in any of the 30 runs. We can see that FRST , FFF and GAUX obtained the optimum on 6 of 9 problems, while GAOP on 5 of 9 and SA just in 1 of 9 (this optimum was reached by all algorithms).

PROBLEM	FRST	FGRED	SA	GAOP	GAUX
Pb5	1.04	.92	6.52	3.37	3.07
Pb7	.94	1.19	4.13	3.83	4.23
weing1	.20	.19	8.07	.92	1.37
weing3	1.54	1.32	22.04	1.85	.91
weing7	.50	.51	.48	1.13	.93
weish10	.27	.14	1.22	1.34	1.18
weish14	.85	.78	1.93	1.85	.91
weish18	.73	.71	1.39	.95	.89
weish27	3.02	2.89	2.85	3.21	1.18
Total	1.01	.96	5.40	2.05	1.63

Table 1 (a)

It is hard to determine why FANS failed to achieve the optimums for those problems, because it is not clear what makes an instance easy or hard. One aspect is the correlation between profits and weights, which is well stablished for classical KP but not for multiple restrictions KP. Other elements are needed but this discussion is out of the aim of this work.

The last row in Table 1 (b), indicates the number of executions ending at the optimum for a total of 9·30=270 runs. FRST and FFF achieves the higher values followed by GAUX. SA and GAOP are quite ineffective from this point of view.

PROBLEM	FRST	FGRED	SA	GAOP	GAUX
Pb5	X	X			
Pb5	X	X			
Pb7	X	X			
Pb7	X	X			
weing1	X	X		X	X
weing1	X	X		X	X
weing3				X	X
weing3				X	X
weing7					
weing7					
weish10	X	X	X	X	X

weish14	X	X		X	X
weish18	X	X		X	X
weish18	X	X		X	X
weish27					X
weish27					X
Total	39	39	9	17	31
Total	39	39	9	17	31

Table 1 (b)

To conclude, taking the mean and variance of the errors over the whole set of problems

ALGOR	Variance	Mean
FRST	1.0024	1.0095
FGRED	1.0244	.9606
SA	7.9356	5.4027
GAOP	1.9580	2.0492
GAUX	1.8684	1.6310

it becomes patent that both version of FANS achieved the lowest values, followed by GAUX. The mean error in GAUX y GAOP was almost twice of that in FRST and FFF; and SA values were 5 times higher.

In order to confirm if mean error differences were of statistical significance, t-tests were done with a confidence level of 95%. The results enabled us to confirm that both versions of FANS outperformed GAUX, GAOP and SA. Both GA outperformed SA, and no significative differences were found among them, in spite of the apparent superiority of GAUX (A + sign at (i,j) means algorithm i was better than j with confidence level of 95\%. A - sign stands for i worst than j and = stands for no significative differences):

	FRST	FGRED	GAUX	GAOP	SA
FRST		=	+	+	+
FGRED	=		+	+	+
GAUX	-	-		=	+
GAOP	-	-	=		+
SA	-	-	-	-	

Acknowledgements

Research supported in part under Projects PB98-1305

Conclusions

In this work we gained evidence about the suitability of fuzy sets based heuristics for solving difficult optimization problems.

Although the main objective was not to develop tools to deal with specific (travelling or knapsack) problems, but to use these problems to learn about fuzzy sets based heuristics

behaviour, the results show the good performace of these algorithms over these problems in spite of the simple component definitions used.

Experiments are being done in order to apply these heuristic algorithms to problems arising in computational molecular biology, where the operator's knowledge is very important to assess the quality of the results.

References

[1] J. Beasley (1997). The or-library: a collection of test data sets. Technical report, Management School, Imperial College, London SW7 2AZ. http://mscmga.ms.ic.ac.uk/info.html.

[2] J.M. Cadenas and J.L. Verdegay (1999): Optimisation Models with Imprecise Data. SPUM (In spanish)

[3] M. Delgado, J. Kacprzyk, J.L. Verdegay and M.A. Vila, Eds (1994): Fuzzy Optimisation: Recent Advances, Physica-Verlag.

[4] A. Diaz, F. Glover, H. Ghaziri, J. Gonzalez, M. Laguna, P. Moscato, and F. Tseng (1996). Heuristic Optimization and Neural Nets. Ed. Paraninfo. In Spanish.

[5] F. Herrera and J.L. Verdegay (1996): Fuzzy Control Rules in Optimisation Problems. Scientia Iranica 3, 89-96.

[6] F. Herrera and J.L. Verdegay (1997): Fuzzy Sets and Operations Research: Perspectives. Fuzzy Sets and Systems 90, 207-218.

[7] J.D.C. Little, K.G. Murty, D.W. Sweeney and C. Karel (1963): An algorithm for the traveling salesman problem, Operations research 11, 972-989.

[8] D. Pelta, A. Blanco, and J. L. Verdegay (2000). A fuzzy adaptive neighborhood search for function optimization. In 4th Int. Conf. on Knowledge-Based Intelligent Engineering Systems & Allied Technologies, KES 2000, vol 2, 594-597.

[9] D. Pelta, A. Blanco, and J. L. Verdegay (2000). Introducing fans: a fuzzy adaptive neighborhood search. In 8th Int. Conf. on Information Processing and Management of Uncertainty in Knowledge-based Systems, IPMU 2000, vol 3, pages 1349-1355.

[10] D.J. Rosenkrantz., R.E. Stearns and P.M. Lewis II (1977): An analysis of several heuristics for the traveling salesman problem. SIAM J. Computing 6, 563-581.

[11] A.P. Sage (1991): Decision Support Systems Engineering. Wiley Series in Systems Engineering. John Wiley and Sons

[12] H.M. Salkin and K. Mathur (1989): Foundations of integer programming, North-Holland, New York.

[13] A. Sancho-Royo, J.L. Verdegay and E. Vergara-Moreno (1999). Some Practical Problems in Fuzzy Sets-based Decision Support Systems. MathWare and Soft Computing VI, 2-3, 173-187.

[14] E.Turban (1988): Decision Support and Expert Systems (Managerial Perspectives). Macmillan Series in Information Systems. Macmillan Publishing Company.

[15] J.L. Verdegay and E. Vergara-Moreno (2000). Fuzzy Termination Criteria in Knapsack Problem Algorithms. Mathware and Soft Computing VII, no. 2-3, 89-97.

[16] J.L. Verdegay y E. Vergara (1999). Fuzzy Stop Criteria for Knapsack Problems. Proceedings of the I Congress of the European Association of Fuzzy Logic and Technologies y

del IX Congreso Español sobre Tecnologias y Logica Fuzzy (1999 EUSFLAT-ESTYLF Joint Conference), Palma de Mallorca, 267-270 (in spanish)

[17] E.R. Vergara (1999): New Termination Criteria for Optimization Algorithms. Ph. D. Dissertation. Universidad de Granada (in spanish).