Using Soft CSPs for Approximating Pareto-Optimal Solution Sets

Marc Torrens and **Boi Faltings**
Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland
Marc.Torrens@epfl.ch, Boi.Faltings@epfl.ch

Abstract

We consider constraint satisfaction problems where solutions must be optimized according to multiple criteria. When the relative importance of different criteria cannot be quantified, there is no single optimal solution, but a possibly very large set of Pareto-optimal solutions. Computing this set completely is in general very costly and often infeasible in practical applications.

We consider several methods that apply algorithms for soft CSP to this problem. We report on experiments, both on random and real problems, that show that such algorithms can compute surprisingly good approximations of the Pareto-optimal set. We also derive variants that further improve the performance.

Introduction

Constraint Satisfaction Problems (CSPs) (Tsang 1993; Kumar 1992) are ubiquitous in applications like configuration, planning, resource allocation, scheduling, timetabling and many others. A CSP is specified by a set of variables and a set of constraints among them. A solution to a CSP is a set of value assignments to all variables such that all constraints are satisfied.

In many applications of constraint satisfaction, the objective is not only to find a solution satisfying the constraints, but also to optimize one or more preference criteria. Such problems occur in resource allocation, scheduling and configuration. As an example, we consider in particular electronic catalogs with configuration functionalities:

- a hard constraint satisfaction problem defines the available product configurations, for example different features of a PC.
- the customer has different preference criteria that need to be optimized, for example price, certain functions, speed, etc.

More precisely, we assume that optimization criteria are modeled as functions that map each solution into a numerical value that indicates to what extent the criterion is violated; i.e. the lower the value, the better the solution.

We call such problems *Multi-criteria Constraint Optimization Problems* (MCOP), which we define formally as follows:

Definition 1. *A Multi-criteria Constraint Optimization Problem (MCOP) is defined by a tuple* $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$, *where* $\mathcal{X} = \{X_1, \ldots, X_n\}$ *is a finite set of variables, each associated with a domain of discrete values* $\mathcal{D} = \{D_1, \ldots, D_n\}$, *and a set of constraints* $\mathcal{C} = \{C_1, \ldots, C_l\}$. *Each constraint* C_i *is defined by a function on some subset of variables. This subset is called the scope of the constraint. A constraint over the variables* $X_1, \cdots, X_k$ *is a function* $D_1 \times \cdots \times D_k \rightarrow \Re^+$ *that defines whether the tuple is allowed (0) or disallowed (*∞*) in case of a hard constraint, or the degree to which it is preferred in the case of a preference criterion, with 0 being the most preferred and* MAX-SOFT[1].

What the best solution to a MCOP is depends strongly on the relative importance of different criteria. This may vary depending on the customer, the time, or the precise values that the criteria take. For example, in travel planning for some people price may be more important than the schedule, while for others it is just the other way around. People find it very difficult to characterize the relative importance of their preferences by numerical weights. Some research has begun to address this problem by inferring constraint weighting from the way people choose solutions (Biso, Rossi, & Sperduti 2000).

Pareto-optimality

When relative importance of criteria is unknown, it is not possible to identify a single best solution, but at least certain solutions can be classified as certainly *not* optimal. This is the case when there is another solution which is as good as or better in all respects. We say that a solution S_1 *dominates* another solution S_2 if for every constraint C_i, the violation cost in S_1 is no greater than that in S_2, and if for at least one constraint, S_1 has a lower cost than S_2. This is defined formally as follows:

Definition 2. *Given a MCOP* P *with constraints* $C =$

[1] `MAX-SOFT` is a maximum value for soft constraints. By using a specific maximum valuation for soft constraints, we can easily differenciate between a hard violation and soft violation.

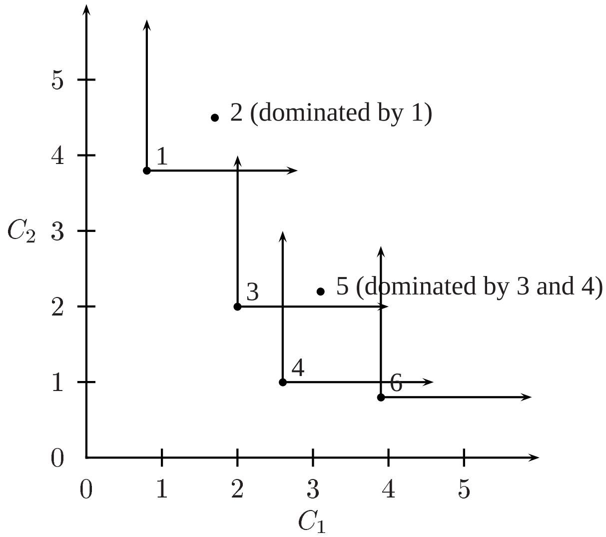

Figure 1: Example of solutions in a CSP with two preference criteria. The two coordinates show the values indicating the degrees to which criteria C_1 (horizontal) and C_2 (vertical) are violated.

$\{C_1, \cdots, C_l\}$ *and two solutions* S_1 *and* S_2 *of* P:

$$S_1 \text{ dominates } S_2 \iff \begin{cases} \forall C_i \in C,\ C_i(S_1) \le C_i(S_2), \text{ and} \\ \exists C_i \in C,\ C_i(S_1) < C_i(S_2) \end{cases}$$

The idea of *Pareto-optimality* (Pareto 1896 1987) is to consider all solutions which are not dominated by another one as potentially optimal:

Definition 3. *Any solution which is not dominated by another is called Pareto-optimal.*

Definition 4. *Given a MCOP P, the Pareto-optimal set* [2] *is the set of solutions which are not dominated by any other one.*

In Figure 1, the Pareto-optimal set is $\{1, 3, 4, 6\}$, as solution 7 is dominated by 4 and 6, 5 is dominated by 3 and 4, and 2 is dominated by 1.

Pareto-optimal solutions are hard to compute because unless preference criteria involve only few of the variables, the dominance relation can not be evaluated on partial solutions. Research on better algorithms for Pareto-optimality is still ongoing (see, for example, Gavanelli (Gavanelli 2002)), but since it cannot escape this fundamental limitation, generating all Pareto-optimal solutions is likely to always remain computationally very hard.

Therefore, Pareto-optimality has so far found little use in practice, despite the fact that it characterizes optimality in a more realistic way. This is especially true when the Pareto-optimal set must be computed very quickly, for example in interactive configuration applications (e.g. electronic catalogs).

Another characteristic of the Pareto-optimal set is that it usually contains many solutions; in fact, all solutions could be Pareto-optimal. Thus, it will be necessary to have the end user, or another program that has the information about relative importance of constraints, pick the best solution among the set that has been returned.

Soft CSPs

Given the intractability of computing all Pareto-optimal solutions, the predominant approach in constraint satisfaction has been to map multiple criteria into a single one and then compute a single solution that is optimal according to this criterion. The key question in this case is how to combine the preference orders of the individual constraints into a global preference order for picking the optimal solution. For the scenario in Figure 1, some commonly used soft CSP algorithms would give the following results:

1. in *MAX-CSP* (Freuder & Wallace 1992), we sum the values returned by each criterion, possibly with a weight, and pick the solution with the lowest sum as the optimal solution. In Figure 1, if we assume that both constraints carry equal weight, this is solution 4.
2. in *fuzzy CSP* (Fargier, Lang, & Schiex 1993), we characterize each solution by the worst criteria violation, i.e. by the maximum value returned by any criterion, and pick the solution with the lowest result as the optimum. In Figure 1, this is solution 3.
3. in *hierarchical CSP* (Borning, Freeman-Benson, & Wilson 1992), criteria have a weight expressing their degree of importance and we order solutions according to the lowest weight of all the constraints that are somehow violated. In Figure 1, if we assume that criterion C_1 is more important than criterion C_2, the optimum is solution 1. If we consider C_2 as more important than C_1, then the optimum is solution 6.

MAX-CSP can be solved efficiently by branch and bound techniques. Fuzzy and hierarchical CSPs can be solved more efficiently by incrementally relaxing constraints in increasing order of importance until solutions are found. More recently, it was observed that most soft CSP methods can be seen as instances of the more general class of soft constraint using c-semirings introduced by (Bistarelli *et al.* 1999). Depending on the way that the semiring operators are instantiated, one obtains the different soft constraint formalisms. Semiring-based CSPs are developed in detail in (Bistarelli 2001).

All these methods make the crucial assumption that violation costs for different criteria are comparable or can be made comparable by known weighting metrics. This assumption is also crucial because it ensures that there is a single optimal solution which can be returned by the algorithm. The weights have a crucial influence on the solution that will be returned: in Figure 1, depending on the relative weights, solutions 1, 4 or 6 could be the optimal ones.

Soft CSP with Multiple Solutions

Interestingly, most decision aids already return not the single optimal solution, but an ordered list of the top-ranked solutions. Thus, web search engines return hundreds of documents deemed to be the best matches to a query, and

[2] also called the efficient frontier of P.

electronic catalogs return a list of possibilities that fit the criteria in decreasing degree. In general, these solutions have been calculated assuming a certain weight distribution among constraints. It appears that listing a multitude of nearly optimal solutions is intended to compensate for the fact that these weights, and thus the optimality criterion, are usually not accurate for the particular user.

For example, in Figure 1, if we assume that constraints have equal weight, the order of solutions would be $4 \rightarrow 3 \rightarrow 1 \rightarrow 6 \rightarrow 5 \rightarrow 2 \rightarrow 7$, and the top four according to this weighting are also the Pareto-optimal ones.

The questions we address in this paper follow from this observation:

- how closely do the top-ranked solutions generated by a scheme with known constraint weights, in particular MAX-CSP, approximate the Pareto-optimal set, and
- can we derive variations that cover this set better while maintaining efficiency?

We have performed experiments in the domain of configuration problems that indicate that MAX-CSP can indeed provide a surprisingly close approximation of the Pareto-optimal set both in real settings and in randomly generated problems, and derive improvements to the methods that could be applied in general settings.

Using Soft CSP Algorithms for Approximating Pareto-optimal Sets

To approximate the set of Pareto-optimal solutions, the simplest solution is to simply map the MCOP into an optimization problem with a single criterion obtained by a fixed weighting of the different criteria, called a weighted constrained optimization problem (WCOP):

Definition 5. *A WCOP is an MCOP with an associated weight vector $W = \{w_1, \cdots, w_n\}$, $w_i > 0$. The optimal solution S to a COP is a tuple that minimizes the valuation function*

$$f(S) = \sum_{i=1}^{n} w_i \cdot C_i(S)$$

The k best solutions to a WCOP is the set of the k solutions with the lowest cost. $f(S)$ is called the valuation of S. We call feasible solutions to a WCOP those solutions which do not violate any hard constraint.

Note that when the weight vector consists of all 1s, WCOP is equivalent to MAX-CSP and is also an instantiation of the semiring CSP framework. WCOP can be solved by branch-and-bound search algorithms. These algorithms can be easily adapted to return not only the best solution, but an ordered list of the k best solutions. In our work we use *Partial Forward Checking* (Freuder & Wallace 1992) (PFC), which is a branch and bound algorithm with propagation.

Pareto-optimality of WCOP Solutions

As mentioned before, in practice it turns out that among the k best solutions to a WCOP, many are also Pareto-optimal. Theorem 1 shows indeed that the optimal solution of a WCOP is always Pareto-optimal, and that furthermore among the k best solutions all those which are not dominated by another one are Pareto-optimal for the whole problem:

Theorem 1. *Let be $B_k(W)$ the set of the best k solutions obtained by optimizing with weights $W = (w_1, \cdots, w_l), w_i > 0$. If $S \in B_k(W)$ and S is not dominated by any $X \in B_k(W), X \neq S$ then S is Pareto-optimal.*

Proof. Assume that S is not Pareto-optimal. Then, there is a solution $Y \notin B_k(W)$ which dominates solution S, and by Definition 2:

$$\forall C_i \in C,\ C_i(Y) \leq C_i(S), \text{ and}$$
$$\exists C_i \in C,\ C_i(Y) < C_i(S)$$

As a consequence, we also have:

$$\sum_{i=1}^{l} w_i \cdot C_i(S) > \sum_{i=1}^{l} w_i \cdot C_i(Y)$$

i.e. Y must be better than S according to the weighted optimization function. But this contradicts the fact that $Y \notin B_k(W)$. □

This justifies the use of soft CSP to find not just one, but a larger set of Pareto-optimal solutions. In particular, by filtering the k best solutions returned by a WCOP algorithm to eliminate the ones which are dominated by another one in the set, we find only solutions which are Pareto-optimal for the entire problem. We can thus bypass the costly step of proving non-dominance on the entire solution set.

The first algorithm is thus to find a subset of the Pareto set by modeling it as a WCOP with a single weight vector, generating the k best solutions, and filtering them to retain only those which are not dominated (Algorithm 1).

Algorithm 1: Method for approximating the Pareto-optimal set of a MCOP using a single WCOP using PFC (Partial Forward Checking).

Input: P : MCOP.
k : the maximal number of solutions to compute.

Output: S: an approximation of the Pareto-optimal set.

$S \leftarrow$ PFC (WCOP (P, $(1, 1, ..., 1)$), k)
$S \leftarrow$ eliminateDominatedSolutions (S)

The Weighted-sums Method

The above method has the weakness that it generates solutions that are optimal with respect to a certain weight vector and thus likely to be very similar to one another. The iterated *weighted-sums* approach, as described for example by Steuer in (Steuer 1986), attempts to overcome this weakness by calling a WCOP method several times with different weight vectors. Each WCOP will give us a different subset of Pareto-optimal solutions, and a good distribution of weight vectors should give us a good approximation of the Pareto-optimal set.

Algorithm 2: Weighted-sums method for approximating the Pareto-optimal set of a MCOP. The m weight vectors W_i are generated to give an adequate distribution of solutions.

Input: P : MCOP.
k : the maximal number of solutions to compute.
$W = \{W_1, \cdots, W_m\}$: W_i is a collection of weight vectors.

Output: S: an approximation of the Pareto-optimal set.

$S \leftarrow$ PFC (WCOP (P, $\{1, \cdots, 1\}$), $k/(m+1)$)
foreach W_i **do**
  $S = S \cup$ PFC (WCOP (P, W_i), $k/(m+1)$)
  $S \leftarrow$ eliminateDominatedSolutions (S)

Basically, the proposed main method (Algorithm 2) consists of performing several runs over WCOPs with different weight vectors for the soft constraints and one run over the vector $\{1, \cdots, 1\}$. The method has two parameters: k, which is the maximal number of solutions to be found, and $W = \{W_1, \cdots, W_m\}$ which is the collection of weight vectors. Algorithm 2 performs $m + 1$ iterations, one for each different WCOP with weight vector W_i, and one for a WCOP with the weight vector $\{1, \cdots, 1\}$. At each iteration, it computes the best $k/(m + 1)$ solutions to the corresponding WCOP. At the end of each iteration, dominated solutions are filtered out, so by Theorem 1, the resulting set of solutions are Pareto-optimal.

Empirical Results on Random Problems

We have tested several different instances and variances of the described methods on randomly generated problems:

Method 1: consits of one search using Algorithm 1.

Method 2: Algorithm 2 with $m + 1$ iterations using m randomly generated weight vectors. In our experiments m varied from 3 to 19 in steps of 2.

Method 3: Algorithm 2 with l iterations, one for each constraint. The iteration for the constraint C_i, is performed with the weight vector $W = \{w_1, \cdots, w_l\}$, where $w_{k=i} = 1$ and $w_{k \neq i} = 100$.

Method 4: Algorithm 2 with $\frac{l*(l-1)}{2}$ iterations, one for each cuple of constraints. The iteration for the constraints C_i and C_j, is performed with the weight vector $W = \{w_1, \cdots, w_l\}$, where $w_{k=i} = 1$, $w_{k=j} = 1$ and $w_{k \neq i} = 100$.

Method 5: Algorithm 2 with $\frac{l*(l-1)}{2} + l$ iterations, one for each constraint and one for each cuple of constraints. This method is a mixed method between method 3 and 4. It takes the vectors from both methods.

Method 6: Using the *Lexicographic Fuzzy* CSP (Dubois, Fargier, & Prade 1996) approach for obtaining the k solutions.

The advantage of working with Method 1 is that it uses standard well-known algorithms. The disadvantage is that it tends to find solutions which are in the same area of the solution space. To increase the diversity of the resulting set of Pareto-optimal solutions, we propose to perform several iterations with random weight vectors (Method 2). The intuition behind method 3 is to remove the *effect* of one criteria at each iteration in order to avoid strong dominance of some of the constraints. The same idea is behind methods 4 and 5. The motivation for method 6 is to compare how well other instantiations of the semiring CSP framework apply to this problem. Lexicographic Fuzzy CSPs are the most powerful version of Fuzzy CSP methods which are interesting because they admit more efficient search algorithms (Bistarelli *et al.* 1999).

Random Soft Constraint Satisfaction Problems Generation

The topology of a random soft CSP is defined by: $< n, m, hc, ht, sc, st, maxV >$, where n is the number of variables in the problem and m the size of their domains. hc is the graph density in percentage for unary and binary hard constraints. ht is the tightness in percentage for disallowed tuples in unary and binary hard constraints. sc and st are the graph density and tightness in percentage for unary and binary soft constraints. Valuations for soft constraints can take values from 0 to $maxV$. For simplicity, hard and soft constraints are separated and we are not considering mixed constraints, therefore $hc + sc \leq 100$. For building random instances of soft CSPs, we choose the variables for each constraint following a uniform probabilistic distribution. In the same way, we choose the tuples in constraints. Valuations for soft tuples are randomly generated between 0 and $maxV$ and valuations for hard tuples are represented by a maximum valuation (∞).

The algorithms have been tested with different set of problems of soft CSPs with 5 or 10 variables and 10 values for each variable. Hard unary/binary constraint density hc has been varied from 20% to 80% in steps of 20, and the tightness for hard constraints ht varies also from 20% to 80% in steps of 20. Soft unary/binary constraint density sc has been varied from 20% to 80% in steps of 10, with tightness fixed at $st = 100$. In the case of 5 variables, in total there could be $5 + (5 * 4)/2 = 15$ soft constraints (5 unary constraints and 10 binary constraints). In the case of 10 variables, in total there could be $10 + (10 * 9)/2 = 55$ soft constraints (10 unary constraints and 45 binary soft constraints).

For every different class of problems, 50 different instances were generated, and each instance has been tested with the different proposed methods. The methods have been tested varying the number of total solutions to be computed from 30 to 530 in steps of 50, from 530 to $2,030$ in steps of 100, from $2,030$ to $10,030$ in steps of $1,000$ and from $10,030$ to $20,030$ in steps of $10,000$.

For the different problem topologies the average of the results for each instance are evaluated in the following section.

Results

Firstly, we are interested in knowing how many Pareto-optimal solutions there are in a problem depending on the

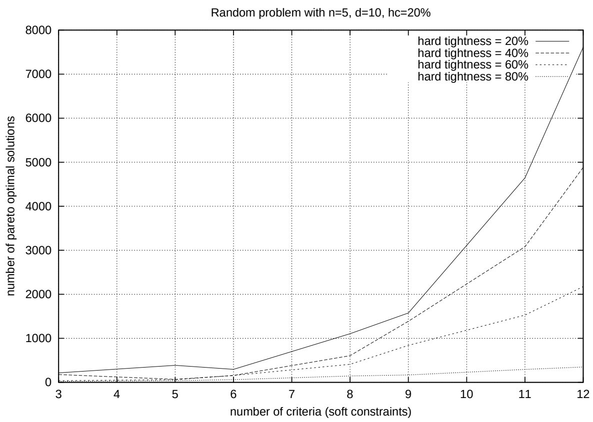

Figure 2: Number of Pareto-optimal solutions depending on how many soft constraints we consider for random generated problems with 5 variables, 10 values per domain and 20% of hard unary/binary constraint density. The number of of solutions in average for the generated problems are: $51,121$ for hard tightness = 20%, $21,792$ for hard tightness = 40%, $6,444$ for hard tightness = 60% and 778 for hard tightness = 80%.

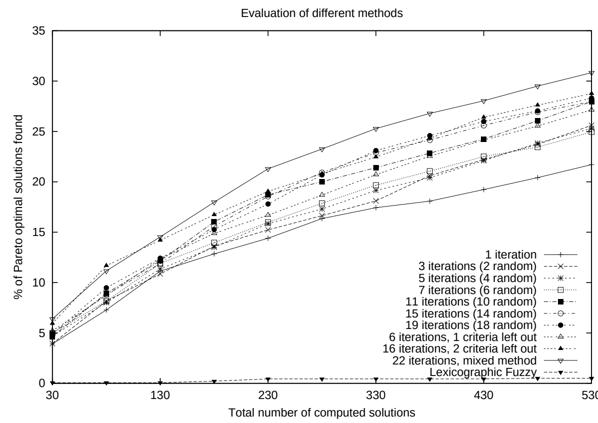

Figure 3: Pareto-optimal solutions found by the different proposed methods (in %). Methods are applied to 50 randomly generated problems with 10 variables, 10 values per domain, 40% of density of hard unary/binary constraints with 40% of hard tightness and 6 criteria (soft constraints). The number of total computed solutions for each method varies from 30 to 530 in steps of 100.

number of criteria (soft constraints). In Figure 2, it is shown that the number of Pareto-optimal solutions clearly increases when the number of criteria increases. The same phenomena applies for instances with 5 and 10 variables. On the other hand, we have observed that even if the number of Pareto-optimal solutions decreases when the problem gets more constrained (less feasible solutions) the percentage in respect to the total number of solutions increases. Thus, the proportion of the Pareto-optimal solutions is more important when the problem gets more constrained.

We have evaluated the proposed methods for each type of generated problems. Figure 3 shows the proportion in avarage of Pareto-optimal solutions found by the different methods for problems with 6 soft constrains. We emphasize the results of the methods up to 530 solutions because in real applications it could not be feasible to compute a larger set of solutions. When computing up to $20,030$ solutions, the behavior of the different methods does not change significantly. The 50 randomly generated problems used for Figure 3 had in avarage $134,661$ feasible solutions (satisfying hard constraints) and 991 Pareto-optimal solutions. The iterative methods perform better than the single search algorithm (Method 1) in respect to the total number of solutions computed. It is worth to note that the iterative methods based on Algorithm 2 find more Pareto-optimal solutions when the number of iterations increase. Lexicographic Fuzzy method (Method 6) results in finding a very low percentage of Pareto-optimal solutions (less than 5%). With Method 6, Theorem 1 does not apply, thus the percentage shown of Pareto-optimal solutions is computed *a posteriori* by filtering out the Pareto-optimal solutions that were not really Pareto-optimal for the entire problem.

Another way of comparing the different methods is to

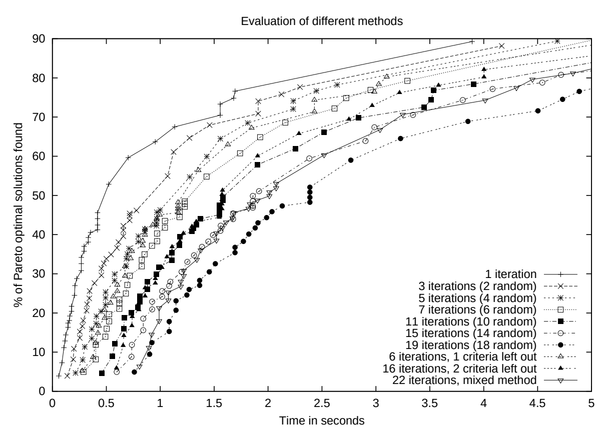

Figure 4: Number of Pareto-optimal solutions found by the different proposed methods with respect to the computing time. For this plot, the problems have 10 variables, 10 values per domain with 40% of hard unary/binary constraints with 40% of hard tightness and 6 criteria (soft constraints).

compare the number of Pareto-optimal solutions found with respect to the computing time (Figure 4). Using this comparison, Method 1 performs the best. The performance of the variants of Method 2 decreases when the number of iterations increases. Method 3 performs better than method 4 which performs better than method 5 in terms of computing time.

In general, we can observe that when the number of iterations of the methods increases the performance regarding the total number of computed solutions also increases but the performance regarding the computing time decreases. This is due to the fact that the computing time of finding the k best solutions with PFC is not linear with respect of finding the k best solutions with m iterations (k/m solutions per iteration). For example, computing $1,030$ solutions with one iteration takes 0.3114 seconds and computing $1,030$ solutions with 7 iterations (of 147 solutions) takes 1.049 seconds.

Even if the tests based on Algorithm 2 takes more time than Algorithm 1 for getting the same percentage of Pareto-optimal solutions, they are likely to produce a more representative set of the Pareto-optimal set.

Using a brute force algorithm that computes all the feasible solutions and filter out those which are dominated took in avarage 9.233 seconds for the same problems as in the above figures. This demonstrates the interest of using approximative methods for computing Pareto-optimal solutions, especially for interactive configuration applications (e.g. electronic catalogs).

Empirical Results in a Real Application

The Air Travel Configurator

The problem of arranging trips is here modeled as a soft CSP (see (Torrens, Faltings, & Pu 2002) for a detailed description of our travel configurator). An itinerary is a set of legs, where each leg is represented by a set of origins, a set of destinations, and a set of possible dates. Leg variables represent the legs of the itinerary and their domains are the possible flights for the associated leg. Another variable represents the set of possible fares[3] applicable to the itinerary. The same itinerary can have several different fares depending on the cabin class, airline, schedule and so on. Usually, for each leg there can be about 60 flights, and for each itinerary, there can be about 40 fares. Therefore, the size of the search space for a round trip is $40 * 60^2 = 144,000$ and for a three leg trip is $40 * 60^3 = 8,640,000$. Constraint satisfaction techniques are well-suited for modeling the travel planning problem. In our model, two types of configuration constraints (hard constraints) guarantee that:

1. a flight f_i for a leg i arrives before a flight f_{i+1} for a leg $i+1$ takes off, and
2. a fare can be really applicable to a set of flights (depending on the fare rules).

Users normally have preferences about the itinerary they are willing to plan. They can have preferences about the schedule of the itinerary, the airlines, the class of service,

[3] In travel industry, the fare applicable to an itinerary is not the sum over the fares for each flight.

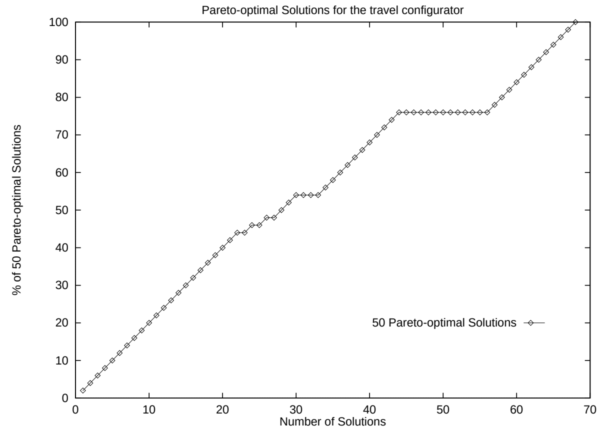

Figure 5: How many solutions we need for getting a certain number of Pareto-optimal solutions ? This example is based on a round trip and shows that, for instance, 50 Pareto-optimal solutions can be found out of about less than 70 solutions.

and so on. Such preferences are modeled as soft constraints. Thus, this problem can be naturally modeled as a MCOP with hard constraints for ensuring the feasibility of the solution and soft constraints for taking into account the user's preferences.

Tests on the Travel Configurator Application

Method 1 has been tested with our travel configurator. We have generated 68 instances of itineraries: 58 round trips, 5 3-leg trips, 1 6-leg trip, 3 5-leg trips and 1 7-leg trip. These instances were tested with 5 unary soft constraints simulating user's preferences. For this application, the goal is to find a set of Pareto-optimal solutions to be shown to the user. Thus, the problem is not to find all Pareto-optimal solutions but a relatively small set of Pareto-optimal solutions. In order to achieve this, we have applied branch and bound algorithm with propagation (PFC) to discover how many solutions we need to obtain a certain number of Pareto-optimal solutions. Precisely, we study how many solutions are needed to find 50 Pareto-optimal solutions.

Evaluation on the Travel Configurator

Figure 5 shows the test results for a round trip (3 variables, with domain sizes 40, 60 and 60) with 5 unary soft constraints (expressing users' preferences). We observe that for getting a certain number of Pareto-optimal solutions in this kind of problems, the number of solutions to compute is very reasonable. Indeed, the method is shown very usable for interactive configuration applications, and specifically for electronic catalogs.

The plot shown in Figure 5 has been generated for all 68 instances of the problems previously described. For all the examples we get similar results. By computing 90 solutions to these problems, we always get 50 Pareto-optimal solutions for all the examples tried.

In electronic catalogs and similar applications, it is useful to find a certain number of Pareto-optimal solutions, even if this set only represents a small fraction of all the Pareto-optimal solutions. Actually, we consider that the number of total solutions that can be shown to the user must be small because of the limitations of the current graphical user interfaces.

Related Work

The most commonly used approach for solving a Multi-criteria Optimization Problem is to convert a MCOP into several COPs which can be solved using standard mono-criteria optimization techniques. Each COP will give then a Pareto-optimal solution to the problem. Steuer's book (Steuer 1986) gives a deep study on different ways to translate a MCOP to a set of COPs. The most used strategy is to optimize by one linear function of all criteria with positive weights. The drawback of the method is that some Pareto-optimal solutions cannot be found if the efficient frontier is not concave[4]. Our methods are based on this approach.

Gavanelli (Gavanelli 2002; 2001) addresses the problem of multi-critera optimization in constraint problems directly. His method is based in a branch and bound schema where the Paerto dominance is checked against a set of previously found solutions using Point Quad-Trees. Point Quad-Trees are useful for efficiently bounding the search. However, the algorithm can be very costly if the number of criteria or if the number of Pareto-optimal solutions are high. Gavanelli's method significantly improves the approach of Wassenhove-Geders (Wassenhove & Gelders 1980). The Wassenhove-Geder's method basically consists of performing several search processes, one for each criteria. Each iteration takes the previous solution and tries to improve it by optimizing another criteria. Using this method, each search produces one Pareto-optimal solution, so a lot of search process must be done in order to approximate the Pareto-optimal set.

The *Global Criterion Method* tries to solve a MCOP as a COP where the criteria to optimize is a minimization function of a distance function to an *ideal* solution. The ideal solution is precomputed by optimizing each criteria independently (Salukvadze 1974).

Incomplete methods have also been developed for solving multi-criteria optimization, basically: *genetic algorithms* (Deb 1999) and methods based on *tabu search* (Hansen 1997).

Conclusions

This paper deals with a very well-studied topic, Pareto-optimality in multi-criteria optimization. It has been commonly understood that Pareto-optimality is intractable to compute, and therefore has not been studied further. Instead, many applications have simply mapped multi-criteria search into a single criterion with a particular weighting and returned a list of the k best solutions rather than a single best one. This solution allows leveraging the well-developed framework of soft CSPs to Multi-criteria Optimization Problems.

Our contribution is to have shown empirically that this procedure, if combined with a filtering that eliminates dominated solutions from the results of the optimization procedure, results indeed a surprisingly good approximations of the Pareto-optimal set. Based on this observation, we have shown that the performance can be improved at a small price in cost by running the same procedure with different random vectors.

We have implemented this method with great success in a commercial travel planning tool, and believe that it would apply well to many other applications as well.

[4] in the case that the optimization function is a minimization function, convex if the optimization function is a maximization function.

References

Biso, A.; Rossi, F.; and Sperduti, A. 2000. Experimental results on Learning Soft Constraints. In *7th International Conference on Principles of Knowledge Representation and Reasoning*.

Bistarelli, S.; Fargier, H.; Montanari, U.; Rossi, F.; Schiex, T.; and Verfaillie, G. 1999. Semiring-based CSPs and Valued CSPs: Basic Properties and Comparison. *CONSTRAINTS: an international journal* 4(3).

Bistarelli, S. 2001. *Soft Constraint Solving and Programming: a general framework.* Ph.D. Dissertation, Università degli Studi di Pisa.

Borning, A.; Freeman-Benson, B.; and Wilson, M. 1992. Constraint Hierarchies. *Lisp and Symbolic Computation: An International Journal* 5(3):223–270.

Deb, K. 1999. Multi-objective genetic algorithms: Problem difficulties and construction of test problems. *Evolutionary Computation* 7(3):205–230.

Dubois, D.; Fargier, H.; and Prade, H. 1996. Possibility Theory in Constraint Satisfaction Problems: Handling priority, preference and uncertainty. *Applied Intelligence* 6:287–309.

Fargier, H.; Lang, J.; and Schiex, T. 1993. Selecting Preferred Solutions in Fuzzy Constraint Satisfaction Problems. In *Proceedings of the First European Congres on Fuzzy and Intelligent Technologies*.

Freuder, E. C., and Wallace, R. J. 1992. Partial constraint satisfaction. *Artificial Intelligence* 58(1):21–70.

Gavanelli, M. 2001. Partially ordered constraint optimization problems. In Walsh, T., ed., *Principles and Practice of Constraint Programming, 7th International Conference - CP 2001*, volume 2239 of *Lecture Notes in Computer Science*, 763. Paphos, Cyprus: Springer Verlag.

Gavanelli, M. 2002. An implementation of Pareto optimality in CLP(FD). In Jussien, N., and Laburthe, F., eds., *CP-AI-OR - International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimisation Problems*, 49–64. Le Croisic, France: Ecole des Mines de Nantes.

Hansen, M. P. 1997. Tabu Search in Multiobjective Optimisation : MOTS. In *Proceedings of MCDM'97*.

Kumar, V. 1992. Algorithms for Constraint Satisfaction Problems: A Survey. *AI Magazine* 13(1):32–44.

Pareto, V. 1896-1987. *Cours d'économie politique professé à l'université de Lausanne*, volume 1. Lausanne: F. Rouge.

Salukvadze, M. E. 1974. On the existence of solution in problems of optimization under vector valued criteria. *Journal of Optimization Theory and Applications* 12(2):203–217.

Steuer, R. E. 1986. *Multi Criteria Optimization: Theory, Computation, and Application*. New York: Wiley.

Torrens, M.; Faltings, B.; and Pu, P. 2002. Smartclients: Constraint satisfaction as a paradigm for scaleable intelligent information systems. *CONSTRAINTS: an international journal* 7:49–69.

Tsang, E. 1993. *Foundations of Constraint Satisfaction*. London, UK: Academic Press.

Wassenhove, L. N. V., and Gelders, L. F. 1980. Solving a bicriterion scheduling problem. *European Journal of Operational Research* 4(1):42–48.