Appeared in *Proceedings of the 5th International Conference on Fuzzy Systems—FUZZ-IEEE'96*, volume 1, pages 306–312, New Orleans, September 1996.

A knowledge-base revision tool for the fuzzy constraints-based *FLIP++ scheduling library

Wolfgang Slany
Christian Doppler Laboratory for Expert Systems
Technical University of Vienna (E184/2)
Paniglgasse 16, A-1040 Wien, Austria, Europe
Phone: +43–1–58801–6123 Fax: +43–1–5055304
mailto:wsi@dbai.tuwien.ac.at
http://www.dbai.tuwien.ac.at/staff/slany.html

Abstract

Real world problems are often overconstrained, ill-defined, or tolerate good though suboptimal solutions as long as those are found before a deadline. Freuder called them partial constraint satisfaction problems (PCSPs). Inspired by classic constraint satisfaction methods, he proposed a model to define and study PCSPs. Another path leading to comparable notions and originating from fuzzy decision making was first investigated by Dubois and Prade. To increase modeling capability, the approaches add one or more attributes such as the following ones to constraints: gradual satisfaction, priority, and compromises. However, given a problem, it is usually difficult to choose a priori parameters for these attributes even within one approach. Counterintuitive small changes in the parameters can have chaotic effects on solution rankings. We propose a new method to select a PCSP model corresponding to the problem one expects to solve. The method basically allows to tune parameters consistently. Theoretical analysis and some initial experiments indicate that our method makes PCSP models ready to confront real world problems.

1. Introduction and motivation

Constraints are mathematical objects used to make explicit the logic behind a problem. Constraint satisfaction refers to algorithms that can be used to solve such problems. Constraints are widely used in AI to model real world decision making problems of *design, configuration, classification (e.g., in machine vision or database queries), planning, and scheduling*. Classic constraint satisfaction problems (CSPs) are usually composed of `hard' constraints, i.e., relations that can be either satisfied or not, without intermediate state. A solution must satisfy all constraints of the problem. If a problem has several solutions that could be further differentiated, it is called `underconstrained'. To differentiate between them, the decision maker has to consider additional constraints. If a problem has no solution at all, it is called `overconstrained'. Some constraints must then be `relaxed' to find acceptable solutions. Both cases are common in real world decision making, for example when buying shoes. There can be many fitting offers or none at all, depending on the criteria and objectives considered, e.g., size, color, design, softness, comfortableness, price, money available, appropriateness for a certain situation, friendliness of the shopkeeper, personal mood, available time for shopping, etc. Nevertheless, almost everybody will eventually be able to find one pair of acceptable shoes, more or less consciously optimizing and compromising between everything that could have some influence on the outcome. Additionally, it turns out that some criteria are easily formulated with words, but cannot as easily be put into a formula. It is for example not easy to formalize how well the shoes will match one's clothes. Human language is also vague when specifying that the new shoes should *not be too expensive*, although there is almost no uncertainty about the price that will have to be paid for a certain pair. Freuder [7] termed the class of these problems partial constraint satisfaction problems (PCSPs).

Another motivation for PCSP models stems from complexity theory. Consider for instance scheduling, a natural candidate for partial constraint satisfaction methods. Scheduling is a hard problem both in practice and theory: On the one hand, practical scheduling problems, although highly constrained, are complex due to the number and variety of the constraints themselves, many of which are `soft', e.g., potentially relaxable human preference constraints, rather than `hard' physical constraints. In addition, a `good' schedule often needs to be evaluated against a number of potentially conflicting goals which themselves may not be precisely defined. Use of analytic techniques to solve practical scheduling problems has in the past been limited due to the lack of suitably expressive languages for constraint representation. On the other hand, theoretical scheduling problems, which are concerned with searching for optimal schedules subject to a limited number of constraints, suffer from excessive combinatorial complexity: The general job-shop scheduling problem with n jobs and m machines has an infinite number of feasible solutions since idle times between operations can vary. A *semi-active* schedule minimizes idle times, but the number of possible solutions is still $(n!)^m$. Simply put, the number of feasible schedules grows exponentially along each dimension (machines, tools, orders, etc.). Therefore, evaluating every solution and then choosing the best one is normally not feasible. Additionally, many of the most commonly encountered constraint satisfaction problems, including scheduling, have been proven to belong to the NP-hard problems. Garey and Johnson [9] state regarding such problems that "Certainly the search for an efficient, exact algorithm should be accorded low priority." Therefore, algorithms finding exact solutions are more or less useless since they will normally not scale to general real world problems for time complexity reasons, unless the problem was trivial anyway. However, it is possible to relax one of the two criteria, either exactness or efficiency, in which case the other criterion can be fulfilled in many cases. One suggestion could be to relax the problem somewhat in its unimportant characteristics, i.e., to model a relaxed version that does yield an *acceptable* solution efficiently. This is often sufficient for real world decision making situations.

Inspired by classic constraint satisfaction methods, Freuder and Wallace [7, 8] proposed a model to define and study PCSPs. Another path leading to comparable notions and originating from fuzzy decision making was first investigated by Dubois and Prade, e.g., [5]. Recent publications on the same track include those by Dubois et al. [4], Fargier et al. [6], Guesgen [10], and Slany [15]. To increase modeling capability, all approaches add one or more attributes such as the following ones to constraints: gradual satisfaction, priority, and compromises. Section 2 shortly introduces our approach. Section 3 goes on to outline some important features. We show that, given a problem, it is usually difficult to choose a priori parameters for these attributes. Counterintuitive small changes in the parameters can have chaotic effects on solution rankings. Freuder and Wallace [8] observe that to change parameters in fact means to solve a different problem. In Section 4, we propose a new method to select a PCSP model corresponding to the problem one expects to solve. The method is universal enough to allow both, comparing the suitability of different approaches for a certain problem, as well as tuning of parameters within one approach. Section 5 concludes the work.

2. Fuzzy decision making

As Freuder and Wallace [8] note, the world in the PCSP domain is not just black and white (consistency or inconsistency), but allows shades of gray. Bonissone [2] describes and compares several formal methods that are well-equipped to handle such shades of gray. In particular, methods belonging to the broad field of fuzzy logic have been applied to PCSPs, for instance by the group lead by Dubois and Prade, e.g., [5]. Similar methods inspired by fuzzy logic have for long been applied in operations research type decision making such as scheduling. Other recent publications on the same track include those by Dubois et al. [4], Fargier et al. [6], and Guesgen [10].

Guesgen [10] and Slany [15] have chosen similar approaches. Constraints are redefined to allow partial satisfaction such that:

Definition 1 Fuzzy constraint satisfaction problems (FCSPs) *consist of:*

$$P\langle V, C_\mu, \mathrm{W}, \mathrm{T}, U\rangle$$

where V is a set of variables, U is the set of universes (domains) for each variable in V, C_μ is a set of constraints where each constraint is a membership function μ *from value assignments to $[0,1]$ having an associated weight w_c representing its importance or priority, plus a weighting scheme* W*, i.e., a function that combines a constraint satisfaction degree $\mu(c)$ with w to yield the weighted constraint satisfaction degree $\mu^w(c)$, and* T *is an aggregation function, e.g., a t-norm (see for instance [12] for more details on triangular norms) that, given C_μ, produces a single partial order on value assignments. An instantiation is a* solution *of the PCSP P iff it is a* maximal *element of the partial order* $\mathrm{T}(C_\mu)$.

Note that continuous domains are possible, i.e., U can be composed of infinite sets. Constraints *can* be violated, i.e., a satisfaction degree of zero means that the corresponding instantiation has to be rejected. Through the definition of μ and W, one can of course define a constraint that will never be rejected, i.e., will always contribute to the aggregated score of the PCSP. Dubois et al. [4], Fargier et al. [6], and Guesgen [10] use the minimum operator (which is a t-norm) as their aggregation function T and choose $\max(1-w_c, \mu_c)$ as their function W. Fargier et al. [6] motivate their choice by arguing that this formalism ensures that w_c represents "the degree to which [a constraint] *must* be satisfied [... The number $(1-w_c)$] measures to what extent it is possible to violate [a constraint]." While mathematically attractive, the use of a t-norm type aggregation, however, precludes compensation of even one bad constraint satisfaction. To solve the PCSPs, Dubois et al. [4], Fargier et al. [6], and Guesgen [10] are using backtracking algorithms adapted from classic CSP methods which are similar to the ones introduced by Freuder and Wallace [8].

2.1. The *FLIP++ libraries

To investigate real world PCSPs in the spirit of Definition 1, we have designed and partly implemented a family of C++ libraries: *FLIP++ (pronounce: StarFlipPlusPlus; this refers to the fact that *FLIP++ stands as a regular expression for all names of the individual sub-libraries, and all of the latter are based on FLIP++, which is a C++ library). *FLIP++ is composed of the following layered sub-libraries: FLIP++: a basic fuzzy logic inference processor library. ConFLIP++: a static fuzzy constraint library. DynaFLIP++: a dynamic fuzzy constraint generation library. DomFLIP++: a domain knowledge representation library. OptiFLIP++: a heuristic optimizing library (several heuristics have so far been implemented and tested, namely: a *tabu list min-conflicts repair based hill climbing heuristic*, a *min-conflicts repair based iterative deepening heuristic*, a *min-conflicts repair based random search hill climbing heuristic*, and a *min-conflicts repair based genetic algorithm heuristic*). **CheckFLIP++**: a knowledge-change consistency checker library that also allows fine-tuning the configuration parameters of a problem. InterFLIP++: a userinterface for all other libraries. DocuFLIP++: an on-line documentation available separately for end-users, knowledge-engineers, and programmers, and accessible via the World-Wide-Web as HTML documents. The functionalities provided by ConFLIP++ as well as some of the other libraries have been described in great detail in [15]. Since we let the user choose among a plethora of different operators and strategies, we do not restate them here. In the present paper, we would like to focus on the role played by CheckFLIP++. Its creation was motivated by the difficulties that arose when larger constraint knowledge bases were investigated.

3. Comparing PCSP models

After having introduced some approaches to solve partial constraint satisfaction problems, we proceed to investigate key issues with regard to these PCSP models.

3.1. Complexity issues

We have designed our constraint satisfaction engine *FLIP++ with a real world problem in mind, namely the scheduling of fine grained production in a steelmaking plant. Typically, this involves more than a thousand binary soft and hard constraints and a matching number of variables with continuous domains. The number of constraint checks until a satisfactory solution is found ranges in the several hundred thousands. In view of these numbers, it is clear that complexity issues play here at least as important a role as the one played by good knowledge acquisition tools. As mentioned above, we therefore adopted *repair based* algorithms that have proven to be very efficient strategies for large constraint satisfaction problems. Minton et al. [14] could find solutions in less than four minutes on a Sparc workstation 1 for the *million*-queens problem, while the best general backtracking approach (found in an empirical study by Stone and Stone [16] to be a most-constrained backtracking algorithm) became intractable for $n > 1000$. The graph on the left side in Figure 1 summarizes the collected statistics that averaged over 100 runs for several different n. Minton et al. [14] even found that their repair based method exhibits linear time and space complexity for large n. The *min-conflicts heuristic* combined with a *repair based hill climbing heuristic* specifies that, starting from an initial suboptimal solution, the system attempts to minimize the number of constraint violations after each repair step. Minton et al. [14] showed convincingly that for certain problems, the use of the additional knowledge gained from operating on complete but suboptimal solutions instead of building solutions from scratch as in constructive approaches pays off well. Such repair based heuristics perform orders of magnitude better than traditional backtracking techniques. Though repair based methods can be combined with many general search strategies, they found that hill climbing methods were especially well suited for the problems they investigated. While this result is very nice for a *general* constraint satisfaction technique, the apparently not well-known fact that Abramson and Yung [1] found a constructive method to solve the general n-queens problem with *linear time* complexity (a corresponding graph has been included in Figure 1) should not be left untold. Though this implies that n-queens is not an intractable problem, it shows that general repair based algorithms often attain almost the optimal theoretical complexity, which seems to be untrue for general constructive backtracking algorithms.

The right side of Figure 1 shows some results of the steelmaking scheduling application for the *tabu list min-conflicts repair based hill climbing heuristic* versus the *min-conflicts repair based random search hill climbing heuristic*, using 10 samples each, with a fixed random initialization, versus a *constructive backtracking heuristic*. More statistics and a detailed analysis of the different algorithms can be found in [15]. All repair based heuristics are much faster and yield better results than the constructive approach that was evaluated using the same configuration parameters.

Another point speaking in favor of repair based approaches for PCSPs is that these algorithms do not need to

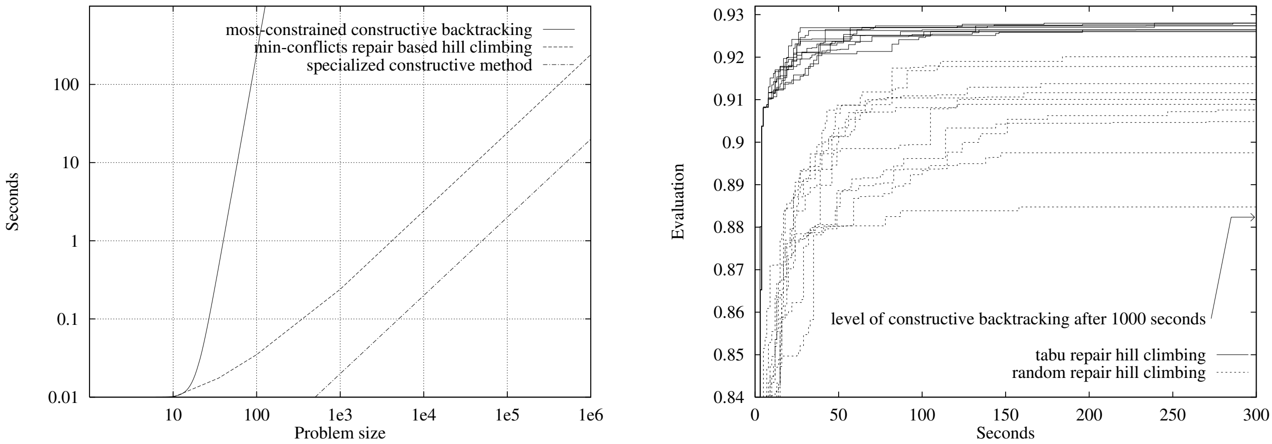

Figure 1. Million-queens statistics with backtracking versus repair versus specialized approach. Steelmaking statistics with two kinds of repair versus one backtracking approach.

prune away search branches and can still be very efficient. While pruning as described in [8] is well suited to optimize classic CSPs, its application to PCSPs poses several problems. For one, compromises can only be evaluated by looking at all constraints. Additionally, real world problems actually often cannot be completely described by constraints, because for instance future events cannot be predicted in scheduling. Therefore, it is sometimes desirable to reject optimal solutions in favor of slightly worse but *robust* solutions, in the sense that small alterations in the actual execution of, e.g., a schedule, still belong to good instantiations, while the superficially best solution is surrounded by very bad ones. By early pruning, it is of course impossible to investigate such a situation.

3.2. Changing PCSP models is chaotic

In [15] we have shown that the ordering behavior of the priority values can be chaotic, in the sense that *small* changes in the knowledge base can have *large* effects on the ranking of solutions. While this seems rather counterintuitive at first, it makes sense after looking closer at the situation. One example in [15] has the ranking of several partial solutions inverted because of unforeseeable interactions between operator fixpoints and weights of constraints, i.e., the rankings of instantiations (large results) are sensitive to certain threshold values in the knowledge base (small changes), so changes can produce unpredictable, chaotic results. In light of the link between nonmonotonic logics and PCSPs as developed by Brewka et al. [3], such nonmonotonic reactions to changes in knowledge bases seem to be an obvious result.

Since there is no unique way to compute weights of constraints for a given problem, there is also no clear way to relate weights to the wishes of the field expert regarding priorities, other than experimenting and fine tuning by testing different variants. It is possible to completely change the ranking behavior of weights by switching to another aggregation operator T or to a different weighting scheme W. The examples given in [15] demonstrate that fine tuning of the parameters for a PCSP is absolutely necessary in order to obtain meaningful results. Section 4 indicates how this tuning can be done rationally while avoiding inconsistencies with former decisions. Since the method is based on trial-and-error, and since test-cases are used to implicitly limit changes in a knowledge base, the method effectively helps to harness the chaotic behavior described above.

4. Managing change in PCSP models

Freuder and Wallace [8] observe that weakening constraints in effect means creating a different problem. In Section 3.2, we have shown that it is often unclear which problem we should solve, and that small changes in parameters describing a PCSP might cause large and unforeseeable changes in the corresponding solutions. Therefore, it seems justified to ask what kind of changes should be allowed and what implications these changes might entail.

4.1. A consistency test

An answer to the problem of making sure that fine tuning is done consistently with earlier decisions is to adopt a consistency test for configuration changes. Such configuration changes could be changes in the priorities between constraints, adopting a new aggregation operator, changing hard barriers, changing membership functions, or changing the logical structure of constraints. Basically, this change together with the test produces a new ranking for a given set of new instantiations, while observing predefined rankings for a set of old reference ranking of pairs of instantiations. The mechanism works such that, if the human expert is dissatisfied with a ranking produced by the system, the human expert can slightly change the weights of some constraints, or the exact form of some membership function (e.g., to specify that a hard barrier is actually located slightly higher), or any other parameter of the problem, such as the aggregation operator used. A consistency test will then check whether the new configuration is consistent with the rankings for a set of reference pairs of instantiations. This is done by applying the new configuration, e.g., the set of new weights, to all the old ordered pairs of instantiations, and by calculating their evaluation scores with this new configuration. If for each reference pair the order between the two reference instantiations remains unchanged, this indicates that the new configuration does not invalidate any previous reference ordering. It is compatible with all decisions made in the past that became reference ranking pairs.

If one reference ranking pair is ranked in the opposite order, this means that either the new configuration is *wrong* and has to be changed again, or that some reference ranking pairs are obsolete and should therefore be removed from the reference ranking pair database. In both cases, an inconsistency among the reference rankings and the new ranking is pointed out. This inconsistency has to be resolved such that the resulting system makes rational, predictable, understandable and self-consistent decisions. The probability that the inconsistency is due to noise in the problem description and should therefore be neglected is zero, since all reference rankings have been generated with the explicit aim to change the configuration in order to give them a certain, new order. An inconsistency can point to earlier errors in configuration changes. Since each change is done under supervision, usually by a human expert, and changes are normally only adopted with the explicit goal to produce a different ordering, the inconsistency can not be attributed to noise. Whether such a decision making behavior can be termed *objective* or *subjective* depends on other factors. However, it is usually possible to lead *several* human experts to agree on a common, undisputed subset of some reference ranking pairs of instantiations, or at least to establish several different sets that correspond to configurations which can be further characterized by and saved for later use under such names as, e.g., for scheduling PCSPs, `risky/cost-cutting', `highest-quality', `observe-temporal-constraints', `standard-mix', etc., indicating their general tendency for decision making. This makes clear that there is no notion of a *best* PCSP in our approach, but that several PCSPs optimizing a solution of a real world problem from slightly different points of view can coexist. The corresponding last configuration is saved together with these reference ranking pairs of instantiations as one knowledge base. Of course, not all intermediate stages have to be stored permanently.

```
boolean     consistency_test(
              current_pool,
              old_configuration,
              new_configuration,
              old_reference_ranking_pair_database) {

  new_reference_ranking_pair_database ←
    new reference_ranking_pair_database;
  new_reference_ranking_pair_database.add(
    reference_ranking_pair(
      search_solution(current_pool, old_configuration),
      search_solution(current_pool, new_configuration)));
  consistency_flag ← true;
  for each reference_ranking_pair(
                best_solution_before_change,
                best_solution_after_change)
  in old_reference_ranking_pair_database {
    if satisfaction(best_solution_before_change,
                    new_configuration)
      ≤ satisfaction(best_solution_after_change,
                     new_configuration)
    then                               // old ordering observed
      new_reference_ranking_pair_database.add(
        reference_ranking_pair(best_solution_before_change,
                               best_solution_after_change));
    else {                             // old ordering violated
      outstream << "Inconsistent ranking "
                << reference_ranking_pair(
                     best_solution_before_change,
                     best_solution_after_change)
                << " (is not added to the new database)."
                << flush;
      consistency_flag ← false;
    }
  }
  new_configuration.save(
    new_reference_ranking_pair_database.save(
      consistency_flag));
  return consistency_flag;
}
```

Table 1. Pseudo-code of a consistency test for configuration changes. The program searches a database of reference ranking pairs to determine and report all those pairs in conflict with the new parameters. It also builds up a new database in case the human expert wants to adopt the change, even if it is inconsistent with some old reference ranking pairs. Thus, a hierarchical tree of variants and extensions of configurations, together with the corresponding database, is built up and named interactively by the human expert. If the consistency test returned true, the new configuration is an extension, otherwise it is a variant of the old one.

This permits modeling the intentions of the human expert with maximal flexibility while ensuring rational and predictable behavior after changes in the configuration.

If the new configuration is adopted, the best solution *before* making the configuration change and the best solution *after* making the configuration change become a new reference ranking pair added to the new database associated with the new configuration. In this pair, the best solution *after* making the configuration change is ranked first, and the best solution *before* making the configuration change is ranked second. All data influencing the overall decision function must be stored together with the pair to be able to apply the resulting new decision function in the old context, given the new configuration. Table 1 lists the consistency test in procedural form.

Human experts can specify implicitly the overall configuration of the constraints by asserting a set of `normal' reference rankings. The easiest way to apply the heuristic that establishes consistent configuration parameters for the constraints is to let the human expert do parameter changes, and to later check them out with the introduced consistency test.

4.2. Automatic knowledge acquisition

Huard and Freuder [11] view constraint knowledge base debugging as a partial constraint satisfaction problem in itself. If the constraint knowledge base is erroneously overconstrained, a change that entails a small number of new solutions is more in keeping with Occam's Razor than one that entails many. However, Huard and Freuder [11] consider only overconstrained networks, while we are interested in finding a PCSP model that approximates as closely as possible the *implicit* problem at hand, thus leading us to move from one PCSP to another instead of moving from an overconstrained CSP to an approximating PCSP. Similar to our approach, Huard and Freuder [11] work in cooperation with a human expert. This permits the human user to interactively play *what-if* games, i.e., allowing the expert to see how decision making behavior evolves as changes are made to the PCSP model. On the one hand, Huard and Freuder [11] allow only *one* constraint to be weakened, while our approach is able to cope with any kind of change. On the other hand, our method so far does not make any suggestions for knowledge change, while the knowledge assistant proposed by Huard and Freuder [11] does. In general, the `inverse' problem of finding an appropriate PCSP model given a certain a priori optimal solution, is extremely difficult because of the multitude of changes that could actually occur. Not without good reason do Huard and Freuder [11] limit changes to only one weakening of one constraint and applying it to rather small problems. The difficulty is that humans easily overlook some constraints, especially when the number of constraints is large and the constraints are only vaguely defined. Therefore, the subjective `better' ranking obtained a priori from a human expert will often objectively not be better than the instantiation found by the system because the human expert forgot some constraints, thus forcing the system to learn suboptimal decision making. Therefore, the fine tuning scenario, where human experts repeatedly change constraint parameters such as weights by hand and then compare the respective best solutions, is much better suited to establishing the best configuration for the problem. This certainly comes from the fact that human expert do often have an intuitive notion of `good' and `bad' schedules without being totally aware why they think so. However, it is an open research problem whether this fine tuning can be fully automated when an *objective, not prune to human error a posteriori meta-evaluation* is used, such as one guided by results of quality evaluations.

According to Freuder and Wallace [8], such meta-constraints, as for instance induced by the consistency test proposed in the present paper, “are reminiscent of the concept hierarchies that provide initial bias in machine learning settings, and indeed it is intriguing to think of the constraint satisfaction process as a form of concept learning, synthesizing a relationship from positive and negative information.”

Future work lies in comparing our work to approaches from knowledge acquisition (e.g., human expert models, cooperative knowledge base tuning), machine learning (e.g., case base reasoning), as well as model based diagnosis (e.g., McIlraith and Reiter [13] study the design of tests whose outcomes confirm or refute a hypothesis).

Huard and Freuder [11] test their knowledge elicitation method on random problems. However, they start from an idealized CSP P that must be approximated; our method is useful to find an unknown P, therefore random PCSPs do not help. We currently believe that our method can only be tested through satisfactory application to real world problems.

5. Conclusion

In this paper, we presented problems as well as solutions appearing in real world decision making that can be represented as partial constraint satisfaction problems (PCSPs) based on fuzzy logic.

The purpose of the present paper was to introduce a new method that allows interactive what-if games for arbitrary decision problems. The latter method is an argument based consistency test with a meta-constraint knowledge base that even allows several experts to agree on parameters of a knowledge base for real world decision making. Through the consistency tested by the method, nonmonotonic changes in knowledge bases of PCSPs can be made more predictable. Our theoretical analysis and some initial experiments indicate that our method makes PCSP models ready to confront real world problems.

References

[1] B. Abramson and M. M. Yung. Construction through decomposition: A linear time algorithm for the n-queens problem. Technical Report CUCS-129-84, Department of Computer Science, Columbia University, 1984. Also, in reduced form, as “Divide and Conquer under Global Constraints: A Solution to the N-Queens Problem.” *Journal of Parallel and Distributed Computing*, 6, 649-662, 1989.

[2] P. P. Bonissone. Reasoning, plausible. In S. C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, volume 2, pages 1307–1322. John Wiley & Sons, Inc., 2nd enlarged and revised edition, 1992.

[3] G. Brewka, H. W. Guesgen, and J. Hertzberg. Constraint relaxation and nonmonotonic reasoning. Technical Report TR-92-002, International Computer Science Institute (ICSI), Berkeley, Calif., 1992. URL: ftp://ftp.icsi.Berkeley-.edu/pub/techreports/1992/tr-92-002.ps.Z.

[4] D. Dubois, H. Fargier, and H. Prade. Propagation and satisfaction of flexible constraints. In R. R. Yager and L. A. Zadeh, editors, *Fuzzy Sets, Neural Networks and Soft Computing*, pages 166–187. Van Nostrand Reinhold, New York, 1994.

[5] D. Dubois and H. Prade. Decision evaluation methods under uncertainty and imprecision. In J. Kacprzyk and M. Fedrizzi, editors, *Combining Fuzzy Imprecision with Probabilistic Uncertainty in Decision Making*, volume 310 of *Lecture Notes in Economical and Mathematical Systems*, pages 48–65. Springer Verlag, 1986.

[6] H. Fargier, J. Lang, and T. Schiex. Selecting preferred solutions in fuzzy constraint satisfaction problems. In H.-J. Zimmermann, editor, *EUFIT'93, First European Congress on Fuzzy and Intelligent Technologies*, pages 1128–1134, Aachen, Germany, Sept. 1993. Augustinus Buchhandlung.

[7] E. C. Freuder. Partial constraint satisfaction. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 278–283, 1989.

[8] E. C. Freuder and R. J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman and Co., 1979.

[10] H. W. Guesgen. A formal framework for weak constraint satisfaction based on fuzzy sets. In *Proceedings of ANZIIS-94*, pages 199–203, Brisbane, Australia, 1994. URL: ftp://-ftp.icsi.Berkeley.edu/pub/techreports/1994/tr-94-026.ps.Z.

[11] S. M. Huard and E. C. Freuder. A debugging assistant for incompletely specified constraint network knowledge bases. *International Journal of Expert Systems: Research and Applications*, 6(4):419–446, 1993.

[12] E. P. Klement. Operations on fuzzy sets and fuzzy numbers related to triangular norms. In *Proceedings of the Eleventh International Symposium on Multiple-Valued Logic*, pages 218–225, Norman, 1981. IEEE, New York.

[13] S. McIlraith and R. Reiter. On tests for hypothetical reasoning. In W. Hamscher, L. Console, and J. de Kleer, editors, *Readings in Model-Based Diagnosis*, pages 89–96. Morgan Kaufmann, 1992.

[14] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992.

[15] W. Slany. Scheduling as a fuzzy multiple criteria optimization problem. *Fuzzy Sets and Systems*, 78:197–222, Mar. 1996. Issue 2. Special Issue on Fuzzy Multiple Criteria Decision Making; URL: ftp://ftp.dbai.tuwien.ac.at/pub/papers/slany/fss96.ps.gz.

[16] H. S. Stone and J. M. Stone. Efficient search techniques – an empirical study of the n-queens problem. *IBM Journal of Research and Development*, 31:464–474, 1987.