Evaluation-Based Semiring Meta-Constraints ⋆

Jerome Kelleher and Barry O'Sullivan
{jt.kelleher|b.osullivan}@cs.ucc.ie

Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland

Abstract. Classical constraint satisfaction problems (CSPs) provide an expressive formalism for modelling and solving many real-world problems. However, classical CSPs prove to be restrictive in any situation where uncertainty, fuzziness, probability, optimisation or partial satisfaction are intrinsic. Soft constraints alleviate many of the restrictions imposed by classical constraint satisfaction. In particular, soft constraints provide a basis for capturing notions such as vagueness, uncertainty and cost in the CSP model. In this paper we focus on the semiring-based approach to soft constraints. We present an overview of soft constraints, and the recent functional formulation of the semiring framework in particular, which also plays a tutorial role in this paper. Furthermore, we present a new evaluation-oriented scheme for implementing meta-constraints, which can be applied to any existing implementation to improve its time and space efficiency.

1 Introduction

Classical constraint satisfaction problems (CSPs) provide an expressive formalism for modelling and solving many real-world problems. CSPs allow us to express *constraints* over variables; these constraints state the allowed combinations of instantiated values for variables. In this way we can declaratively state problems and pass the burden of finding solutions to these problems onto the constraint solver.

However, classical CSPs prove to be restrictive in any problems in which uncertainty, fuzziness, probability, optimisation or partial satisfaction are intrinsic. *Soft* constraints alleviate many of these restrictions imposed by classical constraint satisfaction. We present an overview of the semiring framework for soft constraint satisfaction, which can handle all of the above aspects of softness as well as traditional crisp constraints in a unified and elegant manner.

In this paper we discuss the specification and implementation of semiring *meta-constraints* (constraints which depend on other constraints). We show that implementing meta-constraints using the compilation-based approach is fundamentally flawed and results in any algorithm which utilises these useful abstractions having exponential time and space complexity. We show how these problems can be very simply resolved by instead adopting an *evaluation*-based approach to specifying and evaluating these constraints, which we show to be unrestrictive for the definition of more complex constraint processing algorithms.

⋆ This work has received support from Enterprise Ireland under their Basic Research Grant Scheme (Grant Number SC/02/289).

We advocate the use of semiring meta-constraints to reduce the complexity of defining algorithms to efficiently solve soft constraint problems. Such algorithms have been defined in the system given in [2], which are unfortunately highly inefficient due to the representation of meta-constraints used.

Therefore, the contributions of this paper are as follows:

- We present an overview of soft constraints, and the recent functional formulation of the semiring framework in particular, which is useful for tutorial purposes;
- We present a new evaluation-oriented scheme for implementing meta-constraints. This scheme can be applied to any existing implementation to alleviate problems of excessive space usage - which has concomitant negative implications for the time complexity of constraint processing algorithms.

The paper is organised as follows. Section 2 introduces some of the schemes for solving soft constraint problems over different domains of interest, and shows how these all use different ideas of consistency and satisfaction. Section 3 then shows how the semiring framework of Bistarelli et al. [3–5] can unify many disparate models of constraint satisfaction by using a semiring structure to represent consistency levels and the operations needed to combine and compare those levels. We explain semiring meta-constraints and provide some pedagogical examples of *evaluation*-based meta-constraints. In Section 4 we present a brief discussion of the applicability of local-consistency techniques to soft constraint problems, and of the inherent limitations of any system which utilises these techniques. Section 6 presents our scheme for the implementation of evaluation-based meta-constraints, and Section 7 presents some basic results on the runtime efficiency that can be expected for systematic search using these abstractions over problems of different tightness. Finally, Section 8 summarises the ideas of this paper and hints at possible future lines of work.

2 Soft Constraints

In this section we informally present an overview of some of the different models for soft constraints which can be cast in the semiring framework (for a more formal and complete treatment of this subject the reader is referred to the literature [3, 4]). For each of the models we will note three separate facets of the model: the set of consistency values used in the model, the operation used to combine consistency values to determine a total consistency value, and the operation used to compare consistency values to determine which is 'better', if any.

We use the term 'consistency' (or α-consistency [5]) here to denote the degree to which a particular problem is satisfied, according to whatever criteria specified in that particular model. This is an extension of the idea of consistency from classical constraint satisfaction where a problem is consistent if it contains a solution and inconsistent otherwise. In soft constraints we do not have a simple boolean concept of consistency, which allows us to specify any number of optimisation schemes.

In the following discussion we will use the ideas of variables and constraints. Informally, a variable is any single element of a problem which can be assigned values from a fixed domain. A constraint is then a function over some set of variables which returns a consistency value when evaluated for some instantiation of its variables.

2.1 Vagueness

Fuzzy constraints [7, 15] are a significant extension of classical constraint satisfaction. Fuzzy constraints allow us to deal with imprecision and vagueness in constraint-based reasoning by defining constraints as fuzzy set membership functions. The set of consistency values is then defined as the continuous interval $[0, 1]$, 0 denoting definite exclusion of a value from a fuzzy set, 1 denoting definite inclusion in the fuzzy set and all of the intermediate values representing an object's degree of membership of the fuzzy set in question. This is usually interpreted as the degree to which a given constraint is satisfied by a particular instantiation of its variables. This greatly enriches the expressiveness of the constraint satisfaction paradigm as we can naturally express, for example, user preferences, which are often imprecise and vague, and very rarely crisp.

Combining two fuzzy constraints is then equivalent to finding the intersection of two fuzzy sets. For this reason we use the min function, as this is the method used to define the membership function of the intersection of two fuzzy sets. To compare two consistency values to determine which is better we use the max function, as this allows us to determine which value satisfies constraints the most.

Fuzzy constraints can also model ([1]) Partial Constraint Satisfaction [9] which allows for solutions to be found even when a problem is over constrained in classical constraint satisfaction. Fuzzy constraints can also easily model prioritised constraints [6], in which constraints have associated levels of importance.

2.2 Uncertainty

Probabilistic constraint satisfaction can be used to model uncertainty in constraint-based reasoning. In this context the consistency level returned by evaluating a constraint is interpreted as the probability of the event represented by that instantiation occurring; hence the set of consistency values is drawn from the interval $[0, 1]$. Each constraint is then an independent probability function.

To compute the total probability of two independent events we use the product rule, i.e. $P(A\ and\ B) = P(A) \times P(B)$. Therefore, to combine consistency values in the probabilistic CSP model we use multiplication of reals. The best solution of a probabilistic CSP is obviously the instantiation of the variables with the highest probability. Hence, to compare consistency levels we use the max function.

2.3 Cost

Weighted CSPs are used to model situations when problems have sets of individual weighting functions associated with variables. The result of evaluating a constraint function under an instantiation is then the cost of this instantiation according to that particular function. The overall cost of a complete instantiation of the variables is then computed by summing the individual costs of each cost function. This allows us to declaratively state very complex optimisation problems using a problem decomposition approach.

To find a solution to a problem of this type we wish to find a configuration which minimises the total cost over all functions. The maximum consistency possible represents a cost of zero - the closer that we can get to this ideal the better. The minimum

consistency possible represents an infinite cost, which we can use to model configurations which are either highly undesirable or indeed impossible. Therefore, the set of all possible consistency values is the reals.

This allows us to model optimisation problems in which the minimisation of overall cost (time, money, resources, etc) is paramount. This describes a large class of real-world optimisation problems.

3 Semiring Framework

The semiring framework for constraint satisfaction is based upon the central observation that a semiring (a set together with two operations which satisfy certain properties) is all that is needed to describe many constraint satisfaction schemes. The semiring set provides the levels of consistency, which can be interpreted as cost, degrees of preference, probabilities or any other criteria consistent with the requirements of the framework. The two operations then allow us to combine ($\times$) and to compare ($+$) consistency levels from this set.

In the interest of brevity we will restrict our discussion of the semiring framework under the functional formulation [5] to a brief statement of the basic ideas involved. For a more detailed and rigorous treatment of the subject the reader is referred to the literature [1, 3–5], where many key results pertaining to this framework are proved.

Semirings. A c-semiring (constraint-semiring) is a tuple $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ such that:

- A is the set of all consistency values and $\mathbf{0}, \mathbf{1} \in A$. $\mathbf{0}$ is the lowest consistency value and $\mathbf{1}$ is the highest consistency value;
- $+$, the additive operator, is a closed, commutative, associative and idempotent operation such that $\mathbf{1}$ is its absorbing element and $\mathbf{0}$ is its unit element;
- $\times$, the multiplicative operator, is a closed and associative operation such that $\mathbf{0}$ is its absorbing element, $\mathbf{1}$ is its unit element and $\times$ distributes over $+$.

The c-semirings for some typical instances of the semiring framework are:

- Crisp CSP: $\langle \{false, true\}, \vee, \wedge, false, true\rangle$;
- Fuzzy CSP: $\langle \{x \mid x \in [0, 1]\}, max, min, 0, 1\rangle$;
- Probabilistic CSP: $\langle \{x \mid x \in [0, 1]\}, max, \times, 0, 1\rangle$;
- Weighted CSP: $\langle \mathcal{R}^+, min, +, +\infty, \mathbf{0}\rangle$;
- Set-based CSP: $\langle \wp(A), \cup, \cap, \emptyset, A\rangle$.

Constraint Problems. Given a semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ and an ordered set of variables V over a finite domain D, a *constraint* is a function which, given an assignment $\eta : V \rightarrow D$ of the variables, returns a value of the semiring. By using this notation we define $\mathcal{U} = \eta \rightarrow A$ as the set of all possible constraints that can be built starting from S, D and V.

In this *functional* formulation of the semiring framework each constraint is a function (as defined in [5]) and not a pair (as defined in [3, 4]). Each constraint function involves all the variables in V, but it depends on the assignment of only a finite subset of them. For example, a binary constraint $c_{x,y}$ over variables x and y, is a function

$c_{x,y} : V \to D \to A$, but it depends only on the assignment of variables $\{x, y\} \subseteq V$. We call this subset the *support* of the constraint.

More formally, consider a constraint $c \in \mathcal{U}$. We define its support as $supp(c) = \{v \in V \mid \exists \eta, d_1, d_2 . c\eta[v := d_1] \neq c\eta[v := d_2]\}$, where

$$\eta[v := d]v' = \begin{cases} d & \text{if } v = v', \\ \eta v' & \text{otherwise.} \end{cases}$$

Note that $c\eta[v := d_1]$ means $c\eta'$ where η' is η modified with the assignment $v := d_1$ (that is, the operator $[\,]$ has precedence over application).

A *soft constraint satisfaction problem* is a pair $\langle C, con \rangle$ where $con \subseteq V$ and C is a set of constraints: con is the set of variables of interest for the set of constraints C, which may also concern variables not in con.

3.1 Semiring Meta-Constraints

In this paper we introduce the term *semiring meta-constraints* (or more concisely, meta-constraints) as a convenient means of referring to constraint functions defined over other constraints in the semiring framework. Several classes of meta-constraints have been defined in the literature, including *combination* constraints, *projection* constraints, *solution* constraints and $blevel$ constraints [1, 3–5]. In this paper we will focus on combination and projection meta-constraints as both *solution* and $blevel$ meta-constraints are defined in terms of these primitives.

Combination Meta-Constraints. Given the set $\mathcal{U}$, the combination function $\bigotimes$ is defined as $(\bigotimes C)\eta = \prod_{c \in C} c\eta$. This function takes a set of constraints and returns a combination meta-constraint. This definition is the straightforward extension of the $\otimes$ function [5] to sets of constraints.

Informally, combination meta-constraints represent the constraint which is equivalent to all of the constraints in C combined together. This is a very useful abstraction as it allows us to perform all reasoning over single constraints instead of cumbersome sets of constraints. To evaluate a given combination meta-constraint for an instantiation of the variables η simply involves evaluating all of its constituent constraints under η and combining the individual consistency values using the semiring $\times$ operator.

Projection Meta-Constraints. Given a constraint $c \in \mathcal{U}$ and a variable $v \in supp(c)$, the *projection* function $\Downarrow$ is defined as $(c \Downarrow_{(supp(c)-\{v\})})\eta = \sum_{d \in D} c\eta[v := d]$. This function takes a constraint and a set of variables as parameters and returns the constraint which is equivalent to the original constraint with its support reduced to the specified set of variables.

Informally, projecting a constraint c over the set of variables $(supp(c) - \{v\})$ returns a constraint c' which is equivalent to c with the variable v removed from the support. This is done by evaluating $c\eta[v := d]$ (for the instantiation of interest) for all domain values d in the domain of v, and returning the sum of all of these individual consistency values using the semiring additive operator $+$. Effectively then, the value returned from evaluating $c'\eta$ is the maximum consistency value possible for the instantiation of variables η if we can choose any value for the instantiation of v.

3.2 An Example

In this section we present an example weighted constraint problem. In this problem we have two variables, x and y defined over the domain $D = \{1, 2, 3, 4, 5\}$. As this is a weighted constraint problem we use the semiring $S = \langle \mathcal{R}^+, min, +, +\infty, 0 \rangle$. In a problem of this type we have a set of cost functions defined over the variables of interest; each individual cost function describes the cost of one specific section of a configuration under a particular instantiation of that variable or the cost of instantiations over related variables (non-unary constraints). For simplicity we define a generic cost function $cost(a, n) = (n - a)^2$ to enable us to easily demonstrate the ideas in question. In general however, any arbitrary function can be used to describe costs.

In particular, we will define three constraints denoted c_x, c_y and $c_{x,y}$ defined as follows,

$$\begin{aligned} c_x\eta &= cost(2, x), \\ c_y\eta &= cost(4, y), \\ c_{x,y}\eta &= cost(1, y - x). \end{aligned}$$

Unary constraints c_x and c_y are intended to represent the costs associated with an instantiation deviating from an ideal value. For instance, the ideal value for x according to c_x is 2 and any instantiation where x is not set to this value will be penalised proportional to the square of its distance from this value. Binary constraint $c_{x,y}$ is used to illustrate the idea that we can easily model complex interrelationships between variable instantiations.

The constraint problem in this example is then given by $P = \langle \{c_x, c_y, c_{x,y}\}, \{x, y\} \rangle$. To allow us to demonstrate the ideas of evaluation oriented meta-constraints introduced in this paper we will give examples of combination and projection meta-constraints over this problem.

Combination. In this example we demonstrate the evaluation of a combination meta-constraint. To evaluate a combination meta-constraint for a particular instantiation we must evaluate each of the constituent constraints under the instantiation in question and find the product of these values using the semiring multiplicative operator.

In particular, we demonstrate the evaluation of the combination of the constraints c_x, c_y and $c_{x,y}$ - i.e. $\bigotimes\{c_x, c_y, c_{x,y}\}$ - under the instantiation where x has the value 1 and y has the value 5 ($\eta[x := 1, y := 5]$), i.e.,

$$\begin{aligned} (\bigotimes\{c_x, c_y, c_{x,y}\})\eta[x := 1, y := 5] \; &= \\ c_x\eta[x := 1, y := 5] \; &= \; cost(2, 1) = 1 \\ &\times_s \\ c_y\eta[x := 1, y := 5] \; &= \; cost(4, 5) = 1 \\ &\times_s \\ c_{x,y}\eta[x := 1, y := 5] \; &= \; cost(1, 4) = 9. \end{aligned}$$

As the semiring multiplicative operator in this case is addition over reals, the overall cost associated with this instantiation of the variables $\eta[x := 1, y := 5]$ is 11.

Projection. In this example we demonstrate a projection meta-constraint evaluation. In particular we will demonstrate the evaluation of the projection of the constraint $c_{x,y}$ over the set $\{x\}$, i.e. the meta-constraint where we remove y from the support of $c_{x,y}$. Specifically then, we will evaluate $c_{x,y} \Downarrow_{\{x\}}$ under the instantiation $\eta[x := 1, y := 5]$, i.e., the instantiation where x has the value 1 and y has the value 5.

To evaluate the projection of a constraint over a particular subset of its support for a given instantiation we must evaluate the constraint in question for all domain values of variables which have been removed from the support. We then find the sum of all of the individual consistency values using the semiring additive operator, $+_s$.

In this particular example we are projecting the constraint $c_{x,y}$ over the set $\{x\}$. This means we are removing the variable y from the support of the constraint. When we remove a variable from the support of a constraint we are effectively saying that we are not interested in this variable, and we therefore wish to find highest consistency value possible if we can assign any value from D to this variable. To do this we must evaluate the constraint in question for all possible instantiations of y, i.e.,

$$\begin{aligned}
(c_{x,y} \Downarrow_{\{x\}})\eta[x := 1, y := 5] \; &= \\
c_{x,y}\eta[x := 1, y := 1] \; &= \; cost(1,0) = \; 1 \\
&+_s \\
c_{x,y}\eta[x := 1, y := 2] \; &= \; cost(1,1) = \; 0 \\
&+_s \\
c_{x,y}\eta[x := 1, y := 3] \; &= \; cost(1,2) = \; 1 \\
&+_s \\
c_{x,y}\eta[x := 1, y := 4] \; &= \; cost(1,3) = \; 9 \\
&+_s \\
c_{x,y}\eta[x := 1, y := 5] \; &= \; cost(1,4) = 16.
\end{aligned}$$

As the semiring additive operator for the weighted semiring is the min function over reals, the result of evaluating this constraint is 0.

One important idea illustrated in this example is the concept of the support of a constraint. In this example, the support of $c_{x,y} \Downarrow_{\{x\}}$ is $\{x\}$. This means that this constraint depends only on the assignment of values to variable x. This is demonstrated in the example when we evaluate the constraint $c_{x,y} \Downarrow_{\{x\}}$ under the instantiation $\eta[x := 1, y := 5]$, but we evaluate the constraint that it depends on ($c_{x,y}$) for all instantiations where $x := 1$ and $y := d$.

4 Local Consistency in Soft Constraints

The application of local-consistency techniques has been highly successful in classical constraints. Local-consistency techniques choose sub-problems in which to eliminate local inconsistency and then iterate this elimination in all chosen sub-problems until stability. The most widely used local-consistency techniques are arc-consistency algorithms, in which sub-problems contain only one constraint. These techniques have been used to increase search efficiency in crisp CSPs very successfully.

In [4, 3] the authors explored the applicability of local-consistency techniques to instances of the semiring framework, and found that an extra restriction must be placed

on the framework to allow for the application of these methods[1]. This extra restriction is the requirement that the $\times$ semiring operation be idempotent for local-consistency techniques to be meaningful. Specifically, if the $\times$ operator is not idempotent, then, in general, any local consistency algorithm cannot be guaranteed to:

1. terminate;
2. return a problem which is equivalent to the original one;
3. be independent of non-deterministic choices made during the algorithm.

Requiring the $\times$ operator to be idempotent is a severe limitation on the expressiveness of the semiring framework. The operation $\times : A \times A \rightarrow A$ is idempotent if for all $a \in A$, $a \times a = a$. As the multiplicative operation is used for combining consistency values this is an unnatural restriction, as this disallows the cumulative aggregation of consistency values. For instance, the standard addition (weighted semiring) and multiplication (probabilistic semiring) operations defined over the reals do not satisfy this property.

On the other hand, in [3, 4] the authors showed that the concept of Dynamic Programming can be usefully applied to *any* instance of the semiring framework, without requiring the idempotency of the $\times$ operation. This may be a fruitful direction for research into more efficient soft constraint solving algorithms.

5 Existing Implementations

In this section we discuss the published implementations of the semiring framework. There are a number of issues with these implementations: these range from limitations on the types of semirings that can be handled, to runtime efficiency issues.

5.1 `clp(FD,s)`

In [11] the authors present an extension of the `clp(FD)` [8] system, `clp(FD,s)`. This system provides an efficient means of solving constraint problems defined over a subset of the semirings in the semiring framework. However, no implementation of the combination and projection meta-constraints is provided.

In this system, the authors explicitly restrict the scope of the solver to those semirings in which $\times$ is idempotent, and hence do not support the full generality of the semiring framework. Many of the techniques used to gain efficiency utilise properties only present in semirings where the multiplicative operation is idempotent. This may seem like a reasonable compromise; however, this design decision prevents problems defined over the Probabilistic and Weighted semirings from being solved using this system.

5.2 Soft CHR

In [2] the authors present an implementation of the semiring framework based on CHRs [10]. CHRs allow for the simplification and propagation of constraints and have been

[1] Subject to certain caveats.

$f(x,y)$		
x	y	$x^2 + y^3$
1	1	2
1	2	9
1	3	28
⋮	⋮	⋮
5	5	150

Table 1. Compilation-based function $f(x, y) = x^2 + y^3$ defined over domain $\{1, \ldots, 5\}$.

successfully deployed in dozens of projects to implement various crisp solvers. However, as propagation cannot be applied to instantiations where the multiplicative operation is not idempotent, the usefulness of CHRs is limited in this context.

However, the system does provide several algorithms which can be used over all instances of the semiring framework, including Branch & Bound algorithms with both variable and constraint labeling, as well as a Dynamic Programming search algorithm. Unfortunately, the implementation of meta-constraints in this system severely limits the applicability of these useful algorithms.

In this system all meta-constraints are represented *extensionally* as a list of tuple-consistency pairs, using the compilation-based scheme (see Section 6). Savings in space usage are attained by not storing tuples with consistency of zero. However, in general, a k-ary meta-constraint will require exponential time and space to compile and store. Moreover, many of the more complex operations for this system - such as the dynamic-programming solver - use this operation heavily, ensuring that these operations require exponential time and space also.

In the next section we present a simple method to solve this problem of exponential time and storage. Hopefully, this new method can be integrated into the system, which may allow the useful general purpose algorithms provided in the system be applied to non-trivial problems.

6 A New Approach to Implementing Meta-Constraints

While a large amount of work has been published on the theoretical aspects of soft constraints, apart from the two implementations mentioned in Section 5 very little has been published on the subject of practical implementation of soft constraints. We advocate the use of semiring meta-constraints as a useful abstraction to reduce the difficulty of implementing efficient algorithms to solve soft constraint problems in general.

However, currently meta-constraints are not viable as they are both specified and implemented using a compilation-based approach. By compilation-based we mean that when a meta-constraint function is created a lookup table of all possible input values and their corresponding output results is computed and stored. This approach is extraordinarily wasteful of both computing time and space. For instance, if we had a ternary meta-constraint function over variables with domains of size twenty, we would need to

compile a lookup table with 8000 entries. In general, if we have a compilation based meta-constraint function over a set of variables V with domain D, then we will require a lookup table with $|D|^{|V|}$ entries to fully encode the function. This means we need *exponential* time and space to construct these functions.

For example, consider the function $f(x, y)$ shown in Table 1. In this example we show a function which is composed of two functions over different variables with their respective results added together. This is analogous to a combination meta-constraint, which is a function composed of a number of separate functions over different variables with their results combined together using some simple operation. The variables in this function x and y are defined over the domain $D = \{1, \ldots, 5\}$. Even with this tiny domain, it is necessary to truncate the lookup table which we are using for explanatory purposes.

6.1 The Evaluation-Oriented Approach

A far more economical (and indeed simpler) method of implementing meta-constraint functions is to simply store the original constraint functions that are involved and evaluate these *as required* with the instantiation of interest. This approach is a direct implementation of the definition of the combination function, $(\bigotimes C)\eta = \prod_{c \in C} c\eta$. In this way we can create a new meta-constraint function in constant time and with space linear in the number of constraints involved. This is the *evaluation*-based method of implementing meta-constraints.

One possible criticism of this evaluation-based approach is that there may be situations where we need know the value of all possible instantiations for a particular meta-constraint, and furthermore, we may need to find out the value of a particular instantiation many times. However, these situations are hard to imagine and still do not warrant the *storage* of all possible instantiations. If we wish to find the value of every possible instantiation for a given meta-constraint we can simply iterate through all possible instantiations and evaluate the constraint for that instantiation. If the value of an instantiation will be needed many times, it is the responsibility of the specific algorithm which requires this property to determine if it is worthwhile caching the value, *not* the function which calculates it.

Furthermore, if we make the not-unreasonable assumption that in the majority of constraint processing algorithms that we define, we will want to find the value of the least number of instantiations possible, the compilation scheme is highly undesirable. To sum up, *any* algorithm that we define in terms of compilation based meta-constraints will have exponential time and space complexity, regardless of the semantics of the algorithm itself.

6.2 Combination Evaluation

Combination meta-constraints are an extremely useful abstraction as they allow us to treat a set of constraints as a single constraint. Thus, any reasoning or operations that deal with constraints can be defined over a single constraint as we can refer to any set of constraints by their combination as a single constraint. This simplifies both theoretical and practical work with constraints.

Algorithm 1 CombinationEvaluate(η)

```
a ← 1
for all c ∈ C do
  a ← a × cη
  if a = 0 then
    return 0
  end if
end for
return a
```

Combination is a universal operation in constraint satisfaction. Any form of constraint processing which deals with distinct sets of constraints can all be expressed in terms of this operation. Therefore any improvements we make in the time or space efficiency of this operation will have knock-on effects on any other more sophisticated constraint processing that we peform.

To evaluate a combination meta-constraint defined over the set of constraints C at runtime for a given instantiation η we use Algorithm 1. In this algorithm we simply iterate through all of the constraints in C and evaluate each one under the instantiation in question. To prevent unnecessary computation, we use the fact that $\mathbf{0}$ is the absorbing element of the $\times$ operation. In this way, we know that if any single function evaluates to $\mathbf{0}$ then the entire combination constraint will also evaluate to $\mathbf{0}$ and we can therefore immediately return $\mathbf{0}$.

As this lazy-evaluation leverages the full generality of the semiring framework, it applies to *all* instances. For example, in the crisp semiring, this optimisation reduces to the lazy evaluation of the boolean AND operation; over the fuzzy semiring, it reduces to the lazy evaluation of the min function defined over the interval $[0, 1]$.

7 Experimental Evaluation

In this section we present some results on a prototype branch and bound solver developed using combination meta-constraints. To quantitatively determine the performance implications of using an evaluation-oriented method for representing meta-constraints, combination meta-constraints in this system are implemented using both the compilation and evaluation scheme. Specifically, we tested the number of constraint evaluations required to find the set of best solutions to random soft constraint problems using compilation and evaluation based combination meta-constraints.

To generate random soft constraint problems we follow the methodology adopted in [14], in which random Fuzzy CSPs are generated with four specific properties: the number of variables n, the number of domain values per variable m, the density d and tightness t. The tightness of a problem is defined as the number of instantiations which evaluate to 0 over the total number of instantiations for each binary constraint. The remaining instantiations are then assigned a consistency value from the interval $(0, 1]$, which is randomly generated with a uniform distribution. To ensure that anomalous results are not reported over random problems, we performed 50-fold cross validation

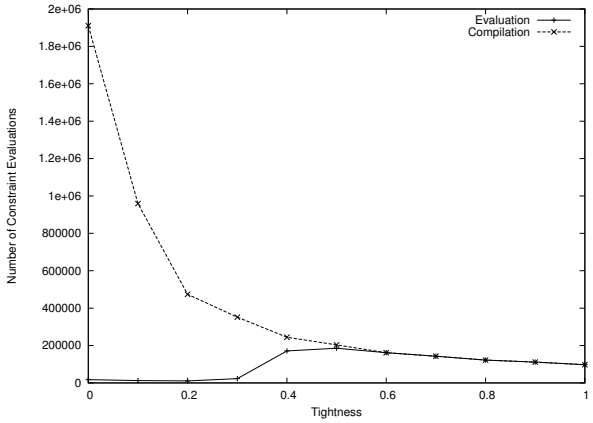

Fig. 1. Branch and Bound search using evaluation-based and compilation-based combination meta-constraints. Random problems have 7 variables, 5 domain values and a density of 1.0.

over results obtained, i.e., we generated 50 random Fuzzy problems with the required specifications and report the average results over these instances.

To implement compilation-based meta-constraints as efficiently as possible we used Algorithm 1 to compute the associated semiring value for each instantiation of the variables in the support of the meta-constraint in question, thus minimising the number of constraint evaluations required. To minimise space requirements, we follow the methodology used in [2] and store only associations between instantiations with non-zero semiring values.

For each problem instance we count the number of constraint evaluations required to find the set of best solutions to the problem using a branch and bound algorithm. This is a vital operation for any soft constraint solver. Figure 1 shows the results of these experiments. Plotted on the x-axis is the problem tightness, or the proportion of instantiations in each constraint which evaluate to zero and on the x-axis the number of constraint evaluations required to find the set of best solutions.

With low tightness values the evaluation oriented approach vastly outperforms the compilation-based approach. This is because we only evaluate constraints on demand in the evaluation-based approach and, because very few constraints evaluate to zero, the bound can be quickly updated and significant branch pruning occurs. In the compilation-based approach however, the entire state-space must be first generated, a value corresponding to the instantiation in question computed, and the association between these two stored *before* any search occurs, resulting in exponential space and time complexity.

At higher levels of problem tightness the performance in terms of constraint evaluations of the two schemes converges. This is because as the difficulty of the problem increases the bound will not be updated and in general we will have to explore a larger proportion of the state-space. The total number of constraint checks required to find the set of best solutions decreases (counter-intuitively) for the compilation-based scheme as tightness increases. This is due to the lazy evaluation given in Algorithm 1, which

allows values to be calculated without requiring all of the constraints to be evaluated when compiling the meta-constraint.

To sum up, the compilation-based approach to implementing meta-constraints is fundamentally flawed as it results in the computation *and* storage of a great deal of information which may not be required to solve a particular problem. The evaluation-based approach is a far simpler and more efficient means of implementing these meta-constraints.

8 Conclusions and Future Work

Classical constraint satisfaction problems (CSPs) provide an expressive formalism for expressing and solving many real-world problems in a declarative fashion. However, classical CSPs prove to be restrictive in any situation where uncertainty, fuzziness, probability, optimisation or partial satisfaction are intrinsic. Soft constraints alleviate many of the restrictions which classical constraint satisfaction imposes. In particular, soft constraints provide a basis for capturing notions such as vagueness, uncertainty and cost into the CSP model.

In this paper we have focused on the semiring-based approach to soft constraints. We presented an overview of soft constraints, and the recent functional formulation of the semiring framework in particular, which also provides a tutorial to the reader who may be unfamiliar with the literature on this subject.

Furthermore, we focused on some critical issues related to the implementation of semiring-based constraint solvers. We presented a new *evaluation-oriented* scheme for implementing meta-constraints, which can be applied to any existing implementation to significantly improve its performance in terms of both space and time complexity.

In future work we intend to investigate the viability of building a general purpose soft constraint solver using an object-oriented language. In particular, we intend to develop an object-oriented soft constraint solver modeled in part after the benchmark crisp constraint system, ILOG Solver [12, 13].

Acknowledgements

We would like to thank Stefano Bistarelli for fruitful discussion and comments.

References

1. S. Bistarelli. *Soft Constraint Solving and programming: a general framework.* PhD thesis, Dipartimento di Informatica, Università di Pisa, Italy, mar 2001. TD-2/01.
2. S. Bistarelli, T. Fruehwirth, M. Marte, and F. Rossi. Soft constraint propagation and solving in constraint handling rules. In *Proc. of the ACM Symp. on Applied Computing*, pages 1–5, 2002.
3. S. Bistarelli, U. Montanari, and F. Rossi. Constraint Solving over Semirings. In *Proc. IJCAI95*, pages 624–630, San Francisco, CA, USA, 1995. Morgan Kaufman.
4. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Solving and Optimization. *Journal of the ACM*, 44(2):201–236, Mar 1997.

5. S. Bistarelli, U. Montanari, and F. Rossi. Soft concurrent constraint programming. In *Proc. ESOP, April 6 - 14, 2002, Grenoble, France*, LNCS, pages 53–67, Heidelberg, Germany, 2002. Springer-Verlag.
6. Alan Borning, Michael Maher, Amy Martindale, and Molly Wilson. Constraint hierarchies and logic programming. In Giorgio Levi and Maurizio Martelli, editors, *Proceedings 6th Intl. Conference on Logic Programming*, pages 149–164. The MIT Press, Cambridge, MA, 1991.
7. James Bowen, Robert Lai, and Dennis Bahler. Lexical imprecision and fuzzy constraint networks. In *Proceedings of AAAI-92*, pages 616–621, July 1992.
8. Philippe Codognet and Daniel Diaz. Compiling constraints in clp(FD). *Journal of Logic Programming*, 27(3):185–226, 1996.
9. Eugene C. Freuder and Richard J. Wallace. Partial Constraint Satisfaction. In *Artificial Intelligence vol. 58*, pages 21–70, 1992.
10. Thom Frühwirth. Theory and practice of constraint handling rules. *Journal of Logic Programming, Special Issue on Constraint Logic Programming*, 37(1-3):95–138, October 1998.
11. Y. Georget and P. Codognet. Compiling semiring-based constraints with `clp(FD,S)`. In *Proc. of CP-98*, LNCS 1520, pages 205–219, 1998.
12. Jean-Francois Puget. A C++ implementation of CLP. In *Proceedings of the Second Singapore International Conference on Intelligent Systems*, Singapore, 1994.
13. Jean-Francois Puget and Michel Leconte. Beyond the glass box: Constraints as objects. In *International Logic Programming Symposium*, pages 513–527, 1995.
14. F. Rossi and I. Pilan. Abstracting soft constraints: Some experimental results. In *Proceedings of the Joint Annual Workshop of the ERCIM Working Group on Constraints and the CoLogNET area on Constraint and Logic Programming*, 2003.
15. Zsofi Ruttkay. Fuzzy constraint satisfaction. In *Proceedings 1st IEEE Conference on Evolutionary Computing*, pages 542–547, Orlando, 1994.