<table>
<thead>
<tr>
<th><strong>Title</strong></th>
<th>Decision diagrams for the computation of semiring valuations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Author(s)</strong></td>
<td>Wilson, Nic</td>
</tr>
<tr>
<td><strong>Publication date</strong></td>
<td>2005-07</td>
</tr>
<tr>
<td><strong>Type of publication</strong></td>
<td>Conference item</td>
</tr>
<tr>
<td><strong>Link to publisher's version</strong></td>
<td><a href="https://www.ijcai.org/Proceedings/2005">https://www.ijcai.org/Proceedings/2005</a></td>
</tr>
<tr>
<td><strong>Rights</strong></td>
<td>© August 1, 2005 International Joint Conferences on Artificial Intelligence. All rights reserved. This publication, or parts thereof, may not be reproduced in any form without permission</td>
</tr>
<tr>
<td><strong>Item downloaded from</strong></td>
<td><a href="http://hdl.handle.net/10468/10768">http://hdl.handle.net/10468/10768</a></td>
</tr>
</tbody>
</table>

Downloaded on 2021-04-29T18:16:15Z
Decision Diagrams for the Computation of Semiring Valuations

Nic Wilson
Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland
n.wilson@4c.ucc.ie

Abstract
This paper describes a new approach to computation in a semiring-based system, which includes semiring-based CSPs (in particular weighted CSPs, fuzzy CSPs and standard CSPs) as well as Bayesian networks. The approach to computation is based on what we call semiring-labelled decision diagrams (SLDDs). These can be generated in a similar way to a standard search tree (decision tree) for solving a CSP, but some nodes are merged, creating a more compact representation; for certain classes of CSPs, the number of nodes in the resulting network will be a tiny fraction of the number of nodes in the corresponding search tree. A method is given for generating an SLDD that represents e.g., a particular instance of a semiring-based CSP; it is shown how this can be used to perform various computations of interest, such as solving a semiring-based CSP, finding optimal solutions, determining the possible values of each variable and counting solutions of a CSP.

1 Introduction
Suppose we are performing a depth-first search for solutions of a (not necessarily binary) CSP, where we record the nodes of the search (decision) tree that we’ve visited so far. Solutions correspond to complete paths in the tree, i.e., directed paths from the root node to a node corresponding to a complete assignment. We are at a node with associated partial assignment \( u \) of set of variables \( U \), which we have checked is consistent (in the CSP sense of it satisfying all constraints involving only variables in \( U \)). Suppose we manage somehow to find another assignment \( u' \) of \( U \), corresponding to a node previously visited, which is also consistent, and which is fully interchangeable, that is (extending [Freuder, 1991]) for all assignments \( w \) of the remaining variables \( W \), \( uw \) is a solution of the CSP if and only if \( u'w \) is a solution.

There is then no need to expand the subtree below \( u \), as it is equivalent to the one below \( u' \) which we have already expanded. Instead we can merge the nodes corresponding to \( u \) and \( u' \). The resulting structure will no longer be a tree—which can be described as a decision diagram (as opposed to a decision tree)—but it will still represent all solutions, as solutions still correspond to complete directed paths; one can also prune the representation to allow backtrack-free generation of any solution. If we can find many such equivalences through interchangeability then the number of nodes in the decision diagram will be a tiny fraction of the number of nodes in the corresponding decision tree. Indeed, the number of solutions can even sometimes be exponential in the size of such a decision diagram, as the merging of nodes ‘factorises’ the solution set (so the size of the decision tree will be exponential in the size of the decision diagram). This kind of representation, in the form of binary decision diagrams [Bryant, 1986; 1995], has proved very effective in certain domains, such as CAD applications.

Testing full interchangeability is often expensive, but one can instead use the sufficient condition, neighbourhood interchangeability, that is, for all the constraints \( c \) whose scope intersects with both \( U \) and \( V - U \), \( c_u = c_{u'} \), where e.g., \( c_u \) (which involves no variables in \( U \)) is the ‘slice’ of \( c \) given by instantiating variables \( U \) to \( u \). This condition can be tested quickly, in time proportional to the size of the (sliced) constraints. Furthermore, if one constructs these slices for each node visited, and indexes them appropriately, then checking whether we have previously generated neighbourhood interchangeable \( u' \), can be achieved efficiently.

This computational approach can be generalised to semiring-based systems (Section 2), including semiring-based CSPs (in particular weighted CSPs, fuzzy CSPs and standard CSPs) as well as Bayesian networks. Each edge in the decision diagram is labelled with a value in the semiring, generating (Section 3) what we call semiring-labelled decision diagrams (SLDDs). We show how this can be done in such a way that the SLDD represents the semiring-based system, i.e., the product of semiring values of edges in a complete path is equal to the semiring value of the complete assignment associated with the path. The efficiency of this approach to computation is very much dependent on how compact the SLDD is, especially compared to the corresponding search tree. If the SLDD is compact, then it can be efficiently constructed and the operations are efficient. We give examples of situations where the SLDD is compact (even linear size) despite the number of solutions being exponential in the number of variables.

Sections 4 and 5 show how the semiring-labelled decision diagram can be used to perform various computations of in-
terest, including solving, optimising and generating inferred constraints, and projections needed in inference in Bayesian networks and solving semiring-based CSPs.

The formalism SLDDs defined in this paper is a development of the finite-state automata representations for weighted constraints in [Amilhastre et al., 2002] (where the soft constraints are unary). The latter, which can be considered as implementing a kind of dynamic programming [Bellman, 1957], has been shown to be applicable as a compilation technique on a substantial real-world problem.

The main contributions of the paper are extending this computational technique of [Amilhastre et al., 2002] in a number of significant ways: our technique, SLDDs, can be applied to computation in many formalisms for preferences and uncertainty: in particular, to a very general class of soft CSPs, semiring-based CSPs; we allow soft constraints to be of arbitrary arity, as well as both extensional and intensional representations of constraints; it applies also to computation in Bayesian networks, and for counting the solutions of a CSP. Our construction of the representation is also new, and is based on backtracking search, where certain nodes in the search tree are merged (implicitly performing a kind of formalised caching); this enables constraint programming techniques to be used, making SLDDs a combination of a form of dynamic programming and search-based approaches used in constraint programming.

2 Semiring Valuations

Semirings consist of a set with two operations [⊙, +] and 0, which are both associative and commutative and such that [⊙] distributes over [+] over [0]. Here we also assume a unit element and a null element. Tuple \( A = \langle A, 0, 1, \odot, \oplus \rangle \) is said to be a semiring if \( A \) is a set containing different elements 0 and 1 and \( \odot \) and \( \oplus \) are operations on \( A \) satisfying the following properties:

- \( \odot \) is associative and commutative with identity 1, \( \odot \) is associative and commutative with identity 0, which is also a null element (i.e., for all \( \alpha \in A \), \( \alpha \odot 0 = 0 \)), and \( \odot \) distributes over \( \oplus \) i.e., for all \( \alpha, \beta, \gamma \in A \), \( \alpha \odot (\beta \oplus \gamma) = (\alpha \odot \beta) \oplus (\alpha \odot \gamma) \).

Semiring valuations. Let \( V \) be a finite set of variables. For each \( X \in V \) let \( D(X) \) be the domain (i.e., the set of possible assignments) of \( X \). For \( U \subseteq V \) let the set of partial tuples \( D(U) = \prod_{X \in U} D(X) \) be the set of possible assignments to set of variables \( U \). A complete assignment is an element of \( D(V) \). For \( u \in D(U) \) and \( W \subseteq U \), \( u^{W} \) is the projection of \( u \) onto variables \( W \), so that for all \( X \in W \), \( u^{W}(X) = u(X) \).

Let \( A = \langle A, 0, 1, \odot, \oplus \rangle \) be a semiring. An \( A \)-valuation \( c \) associates a semiring value with each of a particular set of partial tuples: \( c \) is a function from \( D(V_{c}) \) to \( A \), where set of variables \( V_{c} \subseteq V \) is called the scope of \( c \). If \( u \in D(U) \) is an assignment to set of variables \( U \) containing \( V_{c} \), we may write \( c(u) \) as an abbreviation for \( c(u|V_{c}) \). The semiring operations allow one to define a combination and a projection operation on semiring valuations. The combination \( c \odot c' \) of \( A \)-valuations \( c \) and \( c' \) is defined to have scope \( V_{c} \cup V_{c'} \), and by, \( (c \odot c')(u) = c(u) \odot c'(u) \), i.e., \( c(u|V_{c}) \odot c'(u|V_{c'}') \), for each assignment \( u \) of \( V_{c} \cup V_{c'} \). \( \odot \) is a commutative and associative operation on \( A \)-valuations (on \( V \)). If \( C \) is a multiset of \( A \)-valuations, we may write \( \bigotimes c \) to mean \( \bigotimes_{c \in C} c \). Let \( c \) be an \( A \)-valuation, and let \( U \) be a subset of \( V_{c} \). The projection \( c|U \) of \( c \) onto \( U \) is defined by, for \( u \in D(U) \), \( c|U(u) \) is equal to the semiring summation of \( c(w) \) over all \( w \in D(V_{c}) \) such that \( w|U = u \). Many computational problems can be expressed as computing a projection of a combination \( C = \bigotimes c \) of a multiset of \( C \)-valuations (and a straightforward approach to such a computation is exponential). In particular, computations in semiring-based CSPs [Bistarelli et al., 1997; 1999; 2002] are of this form. This is a general formalism for soft constraints, which includes weighted CSPs, fuzzy CSPs, probabilistic CSPs, lexicographic CSPs and set-based CSPs. These use a special type of semiring, where \( \odot \) is idempotent and 1 is an absorbing element of \( \oplus \).

Finite CSPs can be expressed as a set of \( C \)-valuations, using \( A = \langle \{0, 1\}, 0, 1, \odot, \oplus \rangle \), so that \( \odot = \times \) (i.e., min) and \( \oplus \) is max. A constraint on set of variables \( U \) is represented as a semiring valuation \( c : D(U) \rightarrow \{0, 1\} \), where for assignment \( u \) of \( U \), \( c(u) = 1 \) if and only if \( u \) satisfies the constraint. The CSP has a solution if and only if \( \bigotimes (C) = 1 \).

To perform computations in Bayesian networks, we use the semiring \( A = \langle \mathbb{R}^{+}, 0, 1, +, \times \rangle \), where \( \mathbb{R}^{+} \) is the set of non-negative real numbers. As is well known, computation in Bayesian networks [Pearl, 1988; Shenoy and Shafer, 1990] can be performed by computing projections of combinations of such \( A \)-valuations, where each \( A \)-valuation represents a conditional probability table of a variable given its parents in the Bayesian network. In particular, the marginal probability that variable \( X \) is assigned to \( x \) is given by \( C^{\{X\}}(x) \). This semiring can also be used to count the solutions in a CSP as \( C_{1}^{\{X\}} \), by restricting the input semiring values to being 0 or 1.

3 Constructing an SLDD Representing a Combined Valuation

We will show how to construct a structure, called a semiring-labelled decision diagram (SLDD), that represents, in a particular sense, the combination \( C = \bigotimes c \) of a multiset \( C \) of \( A \)-valuations, where \( A = \langle A, 0, 1, \odot, \oplus \rangle \) is a given semiring. The SLDD is a directed acyclic graph, where the edges are labelled with elements in \( A \). As we’ll show in Sections 4 and 5, this representation can then be used to perform certain important computations.

The construction of the SLDD is very similar to that of a search tree generated by chronological backtracking; the difference is that when we generate an edge we use a certain condition forward neighbourhood interchangeability (defined below, extending the definition of neighbourhood interchangeability in the introduction) to test if we can avoid creating a new node, but instead connect the edge to an already existing node.

Let \( U \subseteq V \) be a set of variables. Define a valuation \( c \in C \) to be active (with respect to \( U \)) if its scope intersects both \( U \) and \( V - U \). For active valuation \( c \), define, for \( u \in D(U) \), \( A \)-valuation \( c_{u} \) to be the ‘slice’ of \( c \) given by assigning \( U \) to
defined the \(A\)-valuation \(c_s\) by, for \(v \in D(V)\), \(c_s(v) = \alpha(\pi_v)\). The SLDD is said to represent \(A\)-valuation \(c_s\).

**Theorem 1** Let \(A\) be a semiring and let \(C\) be a multiset \(C\) of \(A\)-valuations over variables \(V\). Let \(S\) be an \(A\)-decision diagram \(S\) constructed as above. Then \(S\) represents \(C = \bigotimes \sigma C\) and \(C^{10} = \sum_{\pi} \alpha(\pi)\), where the sum is over all complete paths in \(S\).

In the next section it is shown how to compute such sums efficiently.

Note that ‘dynamic variable ordering’ can be used, i.e., the order of variables can differ between paths. We can also adapt constraint programming techniques to improve the efficiency. We can use propagation e.g., maintaining arc consistency for the zeros [Wilson, 2004]. Furthermore, the SLDD can, like a search tree, be generated in a depth-first fashion: this is natural when we’re attempting to solve a CSP; the construction can be terminated when a solution is found.

To facilitate merging, for partial tuple \(u\) we can represent the collection of slices \(\{c_{uv} : c \text{ active}\}\) as a string \(\sigma(u)\) of elements in the semiring. Then, for \(u, u' \in D(U)\), tuples \(u\) and \(u'\) are forward neighbourhood interchangeable if and only if \(\sigma(u) = \sigma(u')\). For set of variables \(U\), we can incrementally construct a trie (a tree of strings) storing each \(\sigma(u_r)\) for all nodes \(r\) with \(u_r \in D(U)\); this enables efficient generation of each edge of the SLDD. For valuations represented extensionally, define \(\text{size}(C)\) to be \(\sum_{c \in C} \text{size}(c)\), where \(\text{size}(c)\) is the cardinality of the scope of \(c\) multiplied by the number of non-zero tups in \(c\). An upper bound for the time needed to construct the SLDD is then proportional to \(\text{size}(C) \times \text{size}(\mathcal{S})\), where \(\text{size}(\mathcal{S})\) is the number of edges of the SLDD. In particular, if the size of the SLDD is polynomial in \(n\), for some parameterised family of problems depending on the number of variables \(n\), then constructing the SLDD is polynomial.

**When is the SLDD compact?**

The crucial factor in determining the efficiency of our approach is the size of the SLDD; if the SLDD is compact, then it can be efficiently constructed and the operations are efficient. The SLDD will tend to be compact (compared to the associated search tree) in situations where many different partial instantiations lead to equivalent subproblems (and so a standard search-based technique would solve the same subproblem many times). There are many kinds of situations where the problem structure causes this to happen.

We give a class of examples to illustrate that the constructed semiring-labelled decision diagram can be compact, even if it represents an exponential number of non-zero complete assignments (e.g., solutions of a CSP). In these cases the size of the SLDD is linear in the number of variables, whereas the size of the corresponding search tree is exponential.

**Example.** Let \(V = \{X_1, \ldots, X_n\}\), where the size of each domain is at most \(d\). Let \(C\) be a multiset of \(A\)-valuations, for some semiring \(A\), such that the scope of each valuation

\[\text{size}(\mathcal{S}) = \bigotimes_{\lambda \in \mathcal{C}} \alpha(\lambda)\]

is equal to the scope of a path \(\pi\) in the decision diagram. This might be thought of as the cost of following path \(\pi\). For \(A\)-decision diagram \(S\)
involves variables which are at most p apart (where p is constant and fairly small): for all \( c \in C \), max \( \{ i : X_i \in V_c \} - \min \{ i : X_i \in V_c \} \leq p \). This might arise, for example, in a problem where there is a temporal component, with the index \( i \) related to time. We can generate an \( A \)-decision diagram using the fixed variable ordering \( X_1, \ldots, X_n \) (i.e., the variables appear in that order in all paths). Let \( j \) be such that \( p < j < n \), and consider any assignments \( u \) and \( u' \) to \( f \) of variables \( U' = \{ X_1, \ldots, X_j \} \). If \( u \) and \( u' \) agree on their last \( p \) variables, i.e., \( u(X_i) = u'(X_i) \) for all \( i \) such that \( j < i < p \), then they are forward neighbourhood interchangeable, since for any active \( A \)-valuation \( c, u \) and \( u' \) agree on the scope of \( c \) and \( u' = u \). This implies that the construction of the SLDD will produce at most \( d^p \) nodes at ‘level’ \( j \), and an upper bound for the size of the SLDD is hence \( nd^{p+1} \); this is linear in \( n \), since \( d \) and \( p \) are constants. For the CSP case, the number of solutions will often be exponential in \( n \), and hence exponential in the size of the SLDD. Many operations, such as counting the number of solutions, or finding an optimal solution for more general semirings (see Section 5) can be achieved in time linear in \( n \), even though an exponential number of assignments are being reasoned about.

A similar argument leads to an upper bound on the size of the SLDD of the same form, but in terms of pathwidth [Bodlaender, 1993], for arbitrary multisets of semiring valuations. (A fixed variable ordering used to generate an SLDD gives rise to a corresponding path decomposition.)

**Other examples.** Certain problems involving permutations can also lead to relatively compact SLDDs; for example, counting the number of linear orders which extend a given partial order [van Dongen, 2004]. The structure of the problem ensures that the SLDD decomposes the problem (similarly to a dynamic programming approach), reducing a factorial problem to an exponential one—which can be a huge reduction in time complexity, but at the cost of large space requirements. Similar remarks apply to solving a rehearsal problem (CSPLib http://www.csplib.org/ problem number 39). Compact structures with a form similar to an SLDD can also be used for enforcing generalised arc consistency for many forms of global constraints e.g., [Pesant, 2004].

### 4 Propagation of Semiring Values in SLDD

The purpose of this section is to show how to efficiently perform various technical computations in an SLDD, in particular, the sum of the semiring values over (i) all complete paths, (ii) all complete paths which pass through a particular node, (iii) all complete paths which pass through a particular edge. The algorithms are immediate generalisations of a classic shortest path algorithm (and the application to the weighted CSP semiring \( 2^V \cup \{ 0, \infty \} \), \( \infty, 0, \text{min}, + \) reduces to this shortest path algorithm).

Suppose we have an \( A \)-decision diagram, with semiring \( A = (A, 0, 1, \oplus, \otimes) \). We associate with each node \( r \) two semiring values \( f(r) \) and \( g(r) \), which are defined inductively. Define \( f(\text{source}) = 0 \), and, working forwards from source, for each \( r \neq \text{source} \), define

\[
  f(r) = \bigoplus_{r' \rightarrow r} (f(r') \otimes g(r' \rightarrow r)).
\]

where the semiring summation is over all edges \( r' \) which point to \( r \). Because the SLDD is a directed acyclic graph with source being the unique parent-less node, this defines \( f(r) \) for each \( r \) unambiguously. Symmetrically, define \( g(\text{sink}) = 1 \), and, working backwards from \( \text{sink} \), define, for \( r \neq \text{sink} \),

\[
  g(r) = \bigoplus_{r' \leftarrow r} (g(r' \leftarrow r) \otimes \alpha(r' \rightarrow r)),
\]

where the semiring summation is over all edges \( r' \) emanating from \( r \). Furthermore, for each node \( r \) and edge \( r \rightarrow r' \), define

\[
  h(r) = f(r) \otimes g(r),
\]

and hence exponential in the size of the SLDD.

**Proposition 1** Let \( r \) be a node and let \( \lambda \) be an edge in the SLDD. If \( r \neq \text{source} \) then \( f(r) = \bigoplus_{\alpha} \alpha(\pi) \) where the semiring summation is over all paths \( \pi \) from \( \text{source} \) to \( r \). In particular, \( f(\text{sink}) \) is the summing \( f(\pi) \) over all paths from \( \text{source} \) to \( \text{sink} \). Similarly, if \( r \neq \text{sink} \),

\[
  g(r) = \bigoplus_{\alpha} \alpha(\pi),
\]

where the semiring summation is over all paths from \( \pi \) to \( \text{sink} \). Also, if \( r \notin \{ \text{source}, \text{sink} \} \),

\[
  h(r) = \bigoplus_{\alpha} \alpha(\pi) \text{ where the semiring summing } \alpha(\pi) \text{ over all complete paths passing through node } r.
\]

Furthermore, \( h(\lambda) \) is equal to the semiring sum of \( \alpha(\pi) \) over all complete paths which include the edge \( \lambda \).

This proposition generalises proposition 8 of [Amilhastre et al., 2002], and the definitions of \( f \) and \( g \) are generalised forms of functions in definition 10 of [Amilhastre et al., 2002].

Note that the number of semiring operations needed to compute functions \( f, g \) and \( h \) is linear in the size of the SLDD.

### 5 Computations Using an SLDD

In this section we show how the SLDD representation can be used to perform important computations: projections and finding optimal solutions. In [Wilson, 2004] we also showed how the SLDD can be used for randomly picking a solution of a CSP with uniform probability (by using an algorithm with a very similar structure to that described below for finding an optimal solution). It is assumed that we have an SLDD representation of a function \( C : D(V) \rightarrow A \), based on a semiring \( A = \langle A, 0, 1, \oplus, \otimes \rangle \); we also assume that we have computed the associated functions \( f, g \), and \( h \), as described in the last section.

#### 5.1 Performing Projections

As mentioned above, computing the projection of the combination of a multiset of semiring valuations is important for many different applications. It gives what is called in [Bistarelli et al., 1997] the ‘solution’ of a semiring-based constraint problem, and can be used for generating implied soft constraints in an idempotent semiring-based CSP [Bistarelli et al., 2002]. Projecting to the empty set of variables can be used, for example, for computing the weight of the best solution in a weighted CSP, or for counting solutions of a CSP. Projection to a singleton set determines the possible values
of a variable for CSPs, and analogously for other semiring-based CSPs; for the Bayesian network semiring it computes a marginal of the Bayesian network.

The lemma below shows how the projection of C onto the empty set and to singleton sets can be computed efficiently. It follows easily from proposition 1.

**Lemma 1** Suppose an A-decision diagram on set of variables V (with associated functions f, g and h) represents A-valuation C. Let X ∈ V be a variable, and x ∈ D(X) be a value of X. Then (i) C^1⊗ = f(sink) = g(source); and (ii) C^1|[X](x) = \bigoplus_{λ∈Λ_x} h(λ), where Λ_x is the set of edges associated with assignment X = x.

So to compute C^1|[X](x) requires |Λ_x| - 1 semiring summations. To compute the projections onto every single variable, i.e., computing C^1|[X](x) for all variables X and all its values x, requires a semiring summation for each edge in the SLDD, so can be computed in time linear in the size of the SLDD.

**Adding unary semiring valuations.** Suppose we receive a new set of semiring valuations, all of which are unary (i.e., the scope of each is a single variable) which we want to combine with the previous combination C. It can be seen that the same graphical structure can be used, but where we just change the semiring values on the edges. This can be used for computing more general projections; it also has applications in in dynamic soft CSPs [Amilhastre et al., 2002], and for conditioning, and hence inference, in a Bayesian network, and computing most probable explanations [Pearl, 1988].

### 5.2 Finding optimal solutions

In this subsection we only consider semirings A = (A, 0, 1, ⊕, ⊗) with a special property: that for all α, β ∈ A, either α ⊕ β = α or α ⊕ β = β. Semirings A satisfying this property are said to satisfy the addition-is-max property. We can then define a total order ≥ on A given by α ≥ β if and only if α ⊕ β = α. Any valuation structure, as defined in definition 11 of [Bistarelli et al., 1999], gives rise to a semiring satisfying addition-is-max, by using the order relation to define ⊕ (note that we are writing the order the opposite way round). Hence the approach in this section applies to any valued CSPs [Bistarelli et al., 1999], including weighted CSPs, fuzzy CSPs and lexicographic CSPs, as valued CSPs can be represented as multisets of A-valuations where A satisfies addition-is-max.

Complete assignment v ∈ D(V) is defined to be optimal if C(v) ≥ C(v') for all v' ∈ D(V). This happens if and only if C(v) = C^1⊗. To generate any optimal complete assignment v ∈ D(V) is very fast: linear in n, the number of variables: we start with node source as the current node, and iteratively pick a child of the current node, satisfying the following condition, until we reach sink: we pick any child r' of current node r with maximum value of α(π)⊗α(r → r')⊗g(r') (among children of r), where π is the path chosen so far, from source to r. The associated complete assignment will be optimal, and conversely, any optimal complete assignment can be generated in this way.

If we are only interested in optimal solutions, we could eliminate each edge λ with h(λ) < C^1⊗ (by connecting such λ to sink and resetting α(λ) to 0), since such an edge is not part of any optimal complete path. For certain addition-is-max semirings (in particular, a fuzzy semiring with ⊗ equalling min, or alternatively, if ⊗ is strictly monotonic), the resulting network has the property that every complete path is optimal (ignoring edges λ with α(λ) = 0). This then gives considerable flexibility in generating optimal solutions in an interactive setting, such as guiding a user in solving a configuration problem.

### 6 Discussion

As well as the representation of [Amilhastre et al., 2002], which they directly generalise (see Section 1), SLDDs are related to several other computational approaches, including search-based approaches for solving CSPs, and join tree/hypertree-based decomposition methods, and AND/OR search graphs. We very briefly discuss these relationships below.

Our construction of the SLDD in Section 3 is different from the way that the representation in [Amilhastre et al., 2002] is generated or similar representations such as Binary Decision Diagrams [Bryant, 1986] and that of [Vempaty, 1992]. The latter involves building up the representation by adding the constraints incrementally, giving representations of subsets of the constraints. SLDDs could also be generated in a similar way; however, the construction in Section 3 takes all the constraints (semiring valuations) into account at once. This can sometimes be advantageous; in particular, when searching for a single solution of a CSP, the SLDD need have time complexity at most polynomially worse than a standard CSP search approach (such as chronological backtracking whilst maintaining arc consistency), whereas the experimental results of [Pan and Vardi, 2004] seem to suggest that a BDD approach can be exponentially worse than a search approach.

SLDDs often have large storage requirements, but they can involve very much fewer nodes (and need never involve more nodes) than a search tree approach. As illustrated by the example in Section 3 (and by the application in [Amilhastre et al., 2002]), certain problems with a very large number of solutions (i.e., non-zero complete assignments) can have an SLDD of manageable size, leading to efficient computation; a search-based method will often then not be feasible if one wants to compile the set of solutions, or count the number of solutions.

Perhaps the most-studied general approach for this kind of problem is the join tree (or hypertree decomposition) approach, e.g., the general framework of Shenoy and Shafer [Shenoy and Shafer, 1990] Bucket Elimination [Dechter, 1999], and non-serial dynamic programming [Bertelet and Briochi, 1972]. Whether a join tree approach or a decision diagram approach is more efficient depends very much on the form of the problem; a join tree approach may be more efficient if there is appropriate topological structure, but little structure which is more value-specific; a decision diagram approach looks liable to be more efficient if there is a good deal of value-specific structure (e.g., of the form of 'context-specific independence' [Boutilier et al., 1996], defined in a Bayesian network context), leading to a compact decision di-
agram. See also discussion in [Wilson, 2004].

An SLDD is similar in form to an OR search graph, a special case of AND/OR search graphs in [Dechter and Mateescu, 2004]; the latter are also closely related to the d-DNNF formalism—see e.g., [Darwiche, 2002], and Case-Factor Diagrams [McAllester et al., 2004]. SLDDs can be extended to also include AND nodes; these can be used when the sliced constraints associated with a node can be partitioned into (e.g., two) sets with non-overlapping scopes; there is no longer a unique final node sink, and the computation generalises that of function g, starting from the sink nodes and working backwards, with semiring product ⊗ being used to combine the semiring values of the branches of an AND node. This sometimes leads to a still more compact representation, but with reduced functionality (as not all the techniques in Section 5 extend).

Summary

This paper introduces and develops semiring-labelled decision diagrams (SLDDs); this computational tool, combining a form of dynamic programming with a constraint programming approach, can be used to solve a variety of important computational problems for (soft or ordinary) constraints and uncertainty. For problem instances with appropriate structure, the SLDD will be compact, and hence be an efficient approach, for example, for solving constraints problems, optimisation, and knowledge compilation.

Acknowledgements

I am grateful for valuable comments from the reviewers (in particular for the pathwidth bound) and Barry O’Sullivan, Stefano Bistarelli, Greg Provan, Marc van Dongen, Gene Freuder, Barbara Smith, Peter Stuckey, Peter MacHale and, especially, Alex Ferguson. This material is based upon work supported by the Science Foundation Ireland under Grant No. 00/PI.1/C075.

References


