Finding Diverse and Similar Solutions in Constraint Programming

Emmanuel Hebrard  
NICTA and UNSW  
Sydney, Australia  
e.hebrard@nicta.com.au

Brahim Hnich  
University College Cork  
Ireland  
brahim@4c.ucc.ie

Barry O’Sullivan  
University College Cork  
Ireland  
b.osullivan@4c.ucc.ie

Toby Walsh  
NICTA and UNSW  
Sydney, Australia  
tw@cse.unsw.edu.au

Abstract

It is useful in a wide range of situations to find solutions which are diverse (or similar) to each other. We therefore define a number of different classes of diversity and similarity problems. For example, what is the most diverse set of solutions of a constraint satisfaction problem with a given cardinality? We first determine the computational complexity of these problems. We then propose a number of practical solution methods, some of which use global constraints for enforcing diversity (or similarity) between solutions. Empirical evaluation on a number of problems show promising results.

Introduction

Computational complexity deals with a variety of different problems including decision problems (e.g. “Is there a solution?”), function problems (e.g. “Return a solution”), and counting problems (e.g. “How many solutions exist?”). However, much less has been said about problems where we wish to find a set of solutions that are diverse or similar to each other. For brevity, we shall call these diversity and similarity problems. Existing work in this area mostly focuses on finding pairs of solutions that satisfy some distance constraint (Bailleux & Marquis 1999; Crescenzi & Rossi 2002; Angelsmark & Thapper 2004). However, there is a wider variety of problems that one may wish to consider.

In product configuration, similarity and diversity problems arise as preferences are elicited and suggestions are presented to the user. Suppose you want to buy a car. There are various constraints on what you want and what is available for sale. For example, you cannot buy a cheap Ferrari nor a convertible with a sunroof. You begin by asking for a set of solutions as diverse as possible. You then pick the most preferred car from this set. However, as not all the details are quite right, you ask to see a set of solutions as similar as possible to this car.

As a second example, suppose you are scheduling staff in a hospital. Unfortunately, the problem is very dynamic and uncertain. People are sure to phone in ill, and unforeseen operations to be required. To ensure that the schedule is robust to such changes and can be repaired with minor disruption, you might look for a schedule which has many similar solutions nearby. Supermodels are based on this idea (Ginsberg, Parkes, & Roy 1998). Finally, suppose you are trying to acquire constraints interactively. You ask the user to look at solutions and propose additional constraints to rule out infeasible or undesirable solutions. To speed up this process, it may help if each solution is as different as possible from previously seen solutions.

In this paper, we introduce several new problems classes, each focused on a similarity or diversity problem associated with a NP-hard problems like constraint satisfaction. We distinguish between offline diversity and similarity problems (where we compute the whole set of solutions at once) and their online counter-part (where we compute solutions incrementally). We determine the computational complexity of these problems, and propose practical solution methods. Finally, we present some promising experimental results.

Formal background

A binary relation $R$ over strings is polynomial-time decidable iff there is a deterministic Turing machine deciding the language $\{x; y \mid (x; y) \in R\}$ in polynomial time (Papadimitriou 1994). A binary relation $R$ is polynomially balanced iff $(x, y) \in R$ implies $|y| < |x|^k$ for some $k$. A language $L$ belongs to NP iff there is a polynomial-time decidable and polynomially balanced relation $R$ such that $L = \{x \mid (x, y) \in R\}$. We let $Sok(x) = \{y \mid (x, y) \in R\}$. We assume that we have some symmetric, reflexive, total and polynomially bounded distance function, $\delta$, between strings. For example, if $L$ is SAT, this might be the Hamming distance between truth assignments. Note, however, that most of our complexity results will hold for any function that is polynomially bounded. Finally, we use 0 to denote $\text{FALSE}$ and 1 to denote $\text{TRUE}$.

Offline diversity and similarity

We define two decision problems at the core of diversity and similarity problems. These ask if there is a subset of solutions of size $k$ at least (or at most) $d$ distance apart. For brevity, we define $\max(\delta, S) = \max_{y, z \in S} \delta(y, z)$ and $\min(\delta, S) = \min_{y, z \in S} \delta(y, z)$.

$d\text{DISTANT}\{\text{Set}\}$ (resp. $d\text{CLOSE}\{\text{Set}\}$)

Instance. Given a polynomial-time decidable and polynomially balanced relation $R$, a distance function
δ and some string x.

**Question.** Does there exist S with S ⊆ Sol(x), |S| = k and min(δ, S) ≥ d (resp. max(δ, S) ≤ d).

We might also consider the average distance between solutions instead of the minimum or maximum. As we have two parameters, d and k, we can choose to fix one and optimize the other. That is, we can fix the size of the set of solutions and maximize their diversity (similarity). Alternatively, we can fix the diversity (similarity) and maximize the size of the set of solutions returned. Alternatively, we could look for a Pareto optimal set or ‘combine d and k into a single metric.

**MaxDiverseSet** (resp. **MaxSimilarSet**)

**Instance.** Given a polynomial-time decidable and polynomially balanced relation R, a distance function δ and some string x.

**Question.** Find S with S ⊆ Sol(x), |S| = k, and for all S’ with S’ ⊆ Sol(x), |S’| = k, min(δ, S) ≥ min(δ, S’) (resp. max(δ, S) ≤ max(δ, S’)).

**MaxDistantSet** (resp. **MaxCloseSet**)

**Instance.** Given a polynomial-time decidable and polynomially balanced relation R, a distance function δ and some string x.

**Question.** Find S with S ⊆ Sol(x), min(δ, S) ≥ d (resp. max(δ, S) ≤ d), and for all S’ with min(δ, S’) ≥ d (resp. max(δ, S’) ≤ d), |S| ≥ |S’|.

For example, in the product configuration problem, we might want to see the 10 most diverse solutions. This is an instance of the **MaxDiverseSet** problem. On the other hand, we might want to see all solutions at most 2 features different to our current best solution. This is an instance of the **MaxCloseSet** problem.

**Online diversity and similarity**

In some situations, we may be computing solutions one by one. For example, suppose we show the user several possible cars to buy but none of them are appealing. We would now like to find a car for sale which is as diverse from these as possible. This suggests the following two online problems in which we find a solution most distant (close) to a set of solutions. For brevity, we define max(δ, S, y) = \max_{z \in S} δ(y, z) and min(δ, S, y) = \min_{z \in S} δ(y, z).

**MostDistant** (resp. **MostClose**)

**Instance.** Given a polynomial-time decidable and polynomially balanced relation R, a distance function δ, some string x and a subset S of strings.

**Question.** Find y with y ∈ Sol(x)−S, such that for all y’ with y’ ∈ Sol(x)−S, max(δ, S, y) ≥ max(δ, S, y’) (resp. min(δ, S, y) ≤ min(δ, S, y’)).

We might also be interested in finding not only but a set of solutions distant from each other and from a given set. Alternatively, we might be interested in finding a larger set of solutions than the ones we currently have with a given diversity or similarity.

**Computational complexity**

For some of these problem classes, specifically those for which the CSP is an input (e.g. dDISTANT/CLOSESET), the size of the output (a set of solutions) may not be polynomially bounded by the size of the input. Therefore we will always make the assumption that k is polynomial in the size of the input. With such an assumption, dDISTANTSet and dCLOSESet are in NP. Since dDISTANTSet and dCLOSESet decide membership in R when k = 1, both problems are trivially NP-complete.

We now show that both **MaxDiverseSet** and **MaxSimilarSet** are FP\(^{NP[log n]}\)-complete. This is the class of all languages decided by a polynomial-time oracle which on input of size n asks a total number of O(log n) NP queries (Papadimitriou 1994).

**Theorem 1** **MaxDiverseSet** and **MaxSimilarSet** are FP\(^{NP[log n]}\)-complete.

**Proof.** **MaxDiverseSet** (resp. **MaxSimilarSet**) ∈ FP\(^{NP[log n]}\): The distance function is polynomial in the size of the input, n. Using binary search, we call dDISTANTSet (resp. dCLOSESet) to determine the exact value of d. This requires only logarithmically many adaptive NP queries.

Completeness: We sketch a reduction from **MaxCliqueSize** which is FP\(^{NP[log n]}\)-complete (Papadimitriou 1994). Given a graph G = (V, E), with set of nodes V and set of edges E, **MaxCliqueSize** determines the size of the largest clique in G. We define a CSP as follows. We introduce a Boolean variable B\(_i\), called a graph-node variable for each node i in V. For each a, b ∈ V, if \{a, b\} ∈ E, we enforce that B\(_a\) and B\(_b\) cannot both be 1. We add an additional \(\frac{n}{2}\) + 1 Boolean variables, called distance variables, and enforce that they are 0 iff every graph-node variable is 0. This CSP always admits the solution with 0 assigned to all Boolean variables. So that **MaxDiverseSet** finds the clique of maximum size, we define the distance between two solutions, δ\(_{s_a, s_b}\), as the Hamming distance defined over all Boolean variables. Thus, **MaxDiverseSet**(P) with k = 2 will always have one of the solutions assigning 0 to each Boolean variable and the other solution assigning 1 to those graph-node variables representing nodes in the largest clique possible, and 0 to all the additional distance variables, since these two solutions are maximally diverse wrt Hamming Distance. Suppose this is not the case, i.e., both solutions assign some node variables to 0 and some others to 1. Since all extra variables are set to 1, the maximum achievable distance is n. Moreover, if the clique with largest cardinality has m nodes, two such solutions cannot be more than 2m assignments apart. However, the solution with maximum clique and the solution with all variables assigned to 0 are \(\lceil \frac{n}{2} \rceil + 1\) assignments apart, which is strictly greater than min(n, 2m). Hence, we have an answer to **MaxCliqueSize(G)**.

The reduction for **MaxSimilarSet** is analogous. However, we need to define the distance between a pair of solutions, s\(_a\) and s\(_b\), as n + \(\lceil \frac{n}{2} \rceil + 1\) − δ\(_{s_a, s_b}\) if s\(_a\) ≠ s\(_b\), and 0 otherwise, where δ is the Hamming distance. □
MaxDistantSet and MaxCloseSet may require exponential time just to write out their answers (since the answer set may itself be exponentially large for certain queries). We therefore assume $d$ (resp. $n - d$) to be at most $O(\log n)$. Furthermore, we assume $\delta$ to be the Hamming distance. These assumptions are sufficient to ensure that the answer set is polynomially bounded and MaxDistantSet (resp. MaxDistantSet) is $FP^{NP[\log n]}$-complete.

**Theorem 2** MaxCloseSet and MaxDistantSet are $FP^{NP[\log n]}$-complete.

**Proof.** MaxCloseSet (resp. MaxDistantSet) is in $FP^{NP[\log n]}$. With the assumptions on $d$ and $\delta$, the cardinality of the set of solutions returned is polynomially bounded. By binary search on $k$, using a $d$CloseSet oracle, we can find the largest possible $d$-close set. Thus, we only need a logarithmic number of calls to an NP oracle to answer both questions. We omit the details of the proof for MaxDistantSet since it is similar ($n - d$ is bounded instead of $d$).

Completeness: We sketch a reduction from MaxClique Size where $G = (V, E)$ denotes a graph with a set of $n$ nodes $V$ and set of edges $E$. We now define a CSP as follows. We introduce $n^2 + 1$ variables. The first variable $X_0$ takes its value in the set $\{1, 2, \ldots, n\}$, and the other variables are partitioned in $n$ groups of $n$ variables, $X_i \in \{0, 1, \ldots, n\}$. We introduce the following constraints:

\[
\forall j, \ X_{(j-1)n+n} = 0 \& (X_0, j) \in E \Rightarrow X_{(X_0 - 1)n + j} = 0, \\
\forall j \neq X_0, k, (k \neq X_0 \& \beta(k, j) \in E) \Rightarrow X_{(k-1)n + j} = X_0.
\]

The entire solution is entailed by the value $i$ assigned to $X_0$, i.e. the $i^{th}$ variable of every group must be assigned to 0, the $j^{th}$ variable of the $i^{th}$ group is assigned to 0 iff $(i, j) \in E$ and to 0 otherwise. Finally, all other variables are set to 0. It is therefore easy to see that we have exactly one solution per node in the graph. Let $\delta(s_a, s_b)$ be the Hamming distance between the solutions $s_a$ and $s_b$. Let $i$ (resp. $j$), be the value assigned to $X_0$ in $s_a$ (resp. $s_b$), we show that $\delta(s_a, s_b) = 2$ iff $(i, j) \in E$. Suppose first that $(i, j) \in E$ and consider the variable $X_{(i-1)n+j}$. In $s_a$ this variable is set to 0 as $(i, j) \in E$, moreover, it is assigned to 0 in $s_b$, as are all $j^{th}$ variables of each group. We can repeat this reasoning for $X_{(j-1)n+i}$, hence $\delta(s_a, s_b) \geq 2$. Now consider any other variable $X_k$. Either $X_k = i$ in $s_a$, or $X_k = j$ in $s_b$, however, no variable is set to $j$ in $s_b$ nor is assigned to $i$ in $s_b$. We thus have $\delta(s_a, s_b) = 2$. Suppose now that $(i, j) \notin E$. The last step in our reasoning is still valid, whilst now we have $X_{(i-1)n+j} = i$ in $s_a$ and $X_{(j-1)n+i} = j$ in $s_b$. Therefore, $\delta(s_a, s_b) = 0$. We first showed that a set of solutions of the CSP map to a set of nodes of $G$, then we showed that if the distance threshold $d$ is set to 2, then this set of nodes is a clique. Since the set with maximum cardinality will be returned, we obtain the solution to MaxClique Size($G$) from MaxCloseSet. The reduction for MaxDistantSet is analogous. However, here we define the distance between a pair of solutions, $s_a$ and $s_b$ as $n^2 + 1 - \delta(s_a, s_b)$, if $s_a \neq s_b$, and 0 otherwise to give a reflexive distance measure.

Finally, we show that the online diversity problem MostDistant and similarity problem MostClose are both $FP^{NP[\log n]}$-complete.

**Theorem 3** MostDistant and MostClose are $FP^{NP[\log n]}$-complete.

**Proof.** MostDistant (resp. MostClose) $\in FP^{NP[\log n]}$: We need to have an algorithm that calls at most a logarithmic number of NP queries. Consider the decision version of MostDistant, where the solution has to have at least $k$ discrepancies with a given set of vectors, $V$. This problem is in NP, the witness being the solution, since checking the number of discrepancies is polynomial. By binary search, we need at most $O(\log n)$ calls to this oracle to determine the most distant solution.

Completeness: We sketch a reduction from MaxClique Size. We define a CSP in the same way as in the completeness proof of Theorem 1 but only use the graph node variables. We let vector $v \in V$ be a string of $0$s of length $n = |V|$. In order that MostDistant finds the clique of maximum size, we define the distance between two solutions, $\delta(s_a, s_b)$ as the Hamming distance. We have an answer to MaxClique Size($G$) iff MostDistant($P, \{v\}$). The reduction for MostClose is analogous, except we define the distance between solutions $s_a$ and $s_b$ as $n - \delta(s_a, s_b)$, if $s_a \neq s_b$, and 0 otherwise to give a reflexive distance measure. We have an answer to MaxClique Size($G$) iff MostClose($P, \{v\}$).

**Solution Methods**


**Complete Methods based on Reformulation**

Given a CSP, we can solve similarity and diversity problems using a reformulation approach. Specifically, we build a new CSP as follows. We create $k$ copies of the CSP, each copy with a different set of variable names. We add $k(k-1)/2$ extra variables to represent the Hamming distance $d^i_j$ between each pair of vectors of variables. We let $\min$ be the minimum of the $d^i_j$’s and $\max$ be the maximum of the $d^i_j$’s. Starting with this basic model, we now describe how to extend it for each of the following problems:

$d$DistanceSet/dCloseSet: We also enforce that $\min$ (resp. $\max$) is at least (resp. at most) $d$.

MaxDiverseSet/MaxSimilkSet: We introduce an objective function that maximizes (resp. minimizes) $\min$ (resp. $\max$).

MostDistant/MostClose: We set $k$ to $|V| + 1$ where $V$ is the input set of solutions and assign $|V|$ sets of variables to the given solutions in $V$. We also introduce an objective function that maximizes (resp. minimizes) $\min$ (resp. $\max$).

We then solve the reformulated problem using off-the-shelf constraint programming solvers. For MaxDistantSet (resp. MaxCloseSet, we find
all solutions and construct a graph whose nodes are the solutions. There is an edge between two nodes iff the pairwise Hamming distance is at least \(d\) (resp. at most \(d\)). We then find a maximum clique in the resulting graph. For problems with many solutions, the constructed graph may be prohibitively large.

**Complete Method for MostDistant/MostClose**

We will show in the next section that the problems MostDistant/MostClose can be used to approximate all the others. Therefore, we introduce here an efficient complete algorithm for MostDistant/MostClose based on global diversity/similarity constraints. We define and study four such global constraints that use Hamming distance. Let \(d\) be an integer, \(X = \{v_1, \ldots, v_k\}\) be a set of vectors of size \(n\), and \(v_j[i]\) be the \(i\)th element of \(v_j\). We define the following global constraints.

\[
\text{Similar}_\Sigma(X_1, \ldots, X_n, V, d) \iff \sum_{i=1}^{n} \sum_{j=1}^{k} (X_i \neq v_j[i]) \leq d
\]

\[
\text{Diverse}_\Sigma(X_1, \ldots, X_n, V, d) \iff \sum_{i=1}^{n} \sum_{j=1}^{k} (X_i \neq v_j[i]) \geq d
\]

\[
\text{Similar}_{\max}(X_1, \ldots, X_n, V, d) \iff \max_{i=1}^{n} \sum_{j=1}^{k} (X_i \neq v_j[i]) \leq d
\]

\[
\text{Diverse}_{\min}(X_1, \ldots, X_n, V, d) \iff \min_{i=1}^{n} \sum_{j=1}^{k} (X_i \neq v_j[i]) \geq d
\]

For reasons of space, we only consider here the diversity constraints. The results are analogous for the similarity ones. We propose a Branch & Bound algorithm for MostDistant that uses these global constraints as follows. We first post a global Diverse constraint with \(d = 1\). For every solution \(s\) that is found during search, we set \(d\) to the Hamming distance between \(s\) and \(s\) plus one, and continue searching. When the algorithm eventually proves unsatisfiability, the last solution found is optimal. We expect propagation on the Diverse constraint to prune parts of the search space.

**Diverse**: A constraint like the global Diverse constraint is generalized arc consistent (GAC), if every value for every variable can be extended to a tuple satisfying the constraint. Consider first the case where there is only one vector \(V\) as argument of Diverse. As long as the number of variables \(X_i\) still containing \(V[i]\) in their domain is greater than \(d\), the constraint is GAC. If this number is less than \(d\), then we fail. Finally, when there are exactly \(d\) such variables, we can remove \(V[i]\) from \(D(X_i)\) for all \(i\).

The situation is more subtle when we look for assignments close to several vectors at once. For instance, consider the following two vectors:

\[
V_1 = \langle 0, 0, \ldots, 0 \rangle, \quad V_2 = \langle 1, 1, \ldots, 1 \rangle
\]

Even if all the domains contain the values 0 and 1, the maximum distance to \(V_1\) and \(V_2\) is \(n\), as any solution has to disagree with either \(V_1\) or \(V_2\) on every variable. Moreover, if we add \(V_3 = \langle 0,1,0, \ldots, 0,1 \rangle\) to this example, then setting a variable with even index to 1 implies that the distance grows by 1, whereas it grows by 2 if we set it to 0, and by 3 if we use any other value. Suppose that all variables take their values in \(\{0,1,2\}\). The assignment with maximum Hamming distance sets to 2 every variable. The distance between this assignment and \(V_1, V_2, V_3\) is \(3n\). Now suppose that \(d = 3n - 1\), then for any \(i\), \(X_i = 0\) and \(X_i = 1\) are arc inconsistent, whilst \(X_i = 2\) is GAC.

We now give an algorithm for enforcing GAC on Diverse\(_\Sigma\)(\(X_1, \ldots, X_n, V_1, \ldots, V_k, d\)).

**Algorithm 1** Diverse\(_\Sigma\)(\(X_1, \ldots, X_n, V_1, \ldots, V_k, d\))

1. \textbf{foreach} \(X_i\) do
2. \quad \textbf{foreach} \(v_j \in D(X_i)\) do
3. \quad \quad \textbf{if} \(v_j \in D(X_i) \land d \leq (\text{Dist} + \text{Best}[i] - \text{Occ}[i][j])\) then
4. \quad \quad \quad \text{Dist} \leftarrow \text{Dist} + k - \min(\text{Occ}[i]);
5. \quad \quad \text{if} \text{Dist} < d \text{ then Fail;}
6. \quad \quad \text{Best}[i] \leftarrow \min(\text{Occ}[i]);
7. \quad \textbf{end foreach}
8. \textbf{end foreach}
9. \textbf{end foreach}

**Theorem 4** Algorithm 1 maintains GAC on Diverse\(_\Sigma\)(\(X_1, \ldots, X_n, V_1, \ldots, V_k, d\)) and runs in \(O(n(d + k))\) where \(d\) is the maximum domain size.

**Proof.** Soundness: suppose that \(v_j \in D(X_i)\) after propagation. We construct an assignment satisfying the constraint, and involving \(X_i = v_j\) as follows: For each variable \(X_j\) other than \(X_i\), we assign this variable to the value \(v_m\) such that Occ\(_i\)[m] is minimum, that is, equal to Best\(_i\). We therefore have \(\sum_{i=1}^{n} \sum_{j=1}^{k} (X_i \neq v_j[x]) = \text{Dist} - \text{Best}[i] + k - \text{Occ}[i][j]\) for this assignment, however, line 3 ensures that this value is greater than \(d\). Hence \(X_i = j\) is GAC.

Completeness: Suppose that \(v_j \not\in D(X_i)\) after propagation. Then we have \(\text{Dist} - \text{Best}[i] + k - \text{Occ}[i][j] < d\), where \(\text{Dist}\) is the sum of Best\(_i\) for all \(i \in [1..n]\). It therefore means that the assignment where every variable but \(X_j\) takes the value occurring the least in \(V_1, \ldots, V_k\), is not a support (a valid extension) for \(X_i = j\). Moreover any other assignment would have a greater distance to \(V_1, \ldots, V_k\), and would therefore not be a support. Hence \(X_i = j\) is not GAC.

**Worst case time complexity:** The loop 1 has complexity \(O(nd + kd)\). The values in Occ\(_i\)[j] can be computed in two passes. The first pass set Occ\(_i\)[j] to 0 for every value \(j\) of every variable \(j\) (\(O(nd)\)). The second pass increments Occ\(_i\)[j] by one for each vector \(V_i\) such that \(V_i[j] = j\) (\(O(nk)\)). The second loop (2) is in \(O(nd)\). 

**Diverse\(_\min\):** Unfortunately, when we use the minimum (resp. maximum) distance, maintaining GAC on a global Diverse (Similar) constraint becomes intractable. We therefore should look to enforce a lesser level of consistency.

**Theorem 5** GAC is NP-hard to propagate on Diverse\(_\min\).

**Proof.** We reduce 3SAT to the problem of finding a consistent assignment for Diverse\(_\min\). Given a
Boolean formula in \( n \) variables \((x_i, \text{ for all } i \in [1..n])\) and \( m \) clauses \((c_i, \text{ for all } i \in [1..m])\), we construct the \( \text{Diverse}_{\min}(X_1, \ldots, X_n, V_1, \ldots, V_m, d) \) constraint in which \( X_i = \{ x_i, \neg x_i \} \) for all \( i \in [1..n] \) and each \( V_j \) for all \( j \in [1..m] \) represents one of the \( m \) clauses. If the clause \( c_i \) is \( \{x_i, \neg x_j, x_k\} \) then \( V_i[i] = 1, V_i[j] = \neg 1 \) and \( V_i[k] \) is \( k \). All other values are set to \( n + 1 \). If \( d = n-2 \), then the constructed \( \text{Diverse}_{\min} \) constraint has a satisfying assignment iff the 3SAT formula has a model. The solution we find corresponds to an assignment of the variables. Moreover, the dummy values “consume” exactly \( n-3 \) differences. Therefore the solution has to share at least one element with each clause, and thus satisfies it. Notice that in this case the solution we find corresponds to a model where all literals are negated.

**Heuristic Approaches**

Methods based on reformulation may give prohibitively large CSPs. We therefore propose an approximation scheme that is based on the Branch & Bound algorithm for MOSTDISTANT/MOSTCLOSE, that approximates the others. Variants of the following greedy heuristic can approximate \( \text{dCLOSE}\text{SEt} \) and \( \text{MaxdCLOSE} \text{Set} \) problem by using the Branch & Bound algorithm for MOSTCLOSE.

**Algorithm 2 greedy**

\[
\text{Data} : P = (X, D, C), K, d
\]
\[
\text{Result} : V
\]

\[
V \leftarrow \{ u \};
\]

\[
\text{while } |V| < K \times \min_{x \in V} \delta(x, y) > d \text{ do}
\]

\[
\text{find most diverse solution } u \text{ to previous solutions } V';
\]

\[
V \leftarrow V \cup \{ u \};
\]

Finding a solution that maximizes similarity to previous solutions (Line 1) corresponds to solving MOSTDISTANT. Observe that if we keep only the first test for the “while” loop, then we approximate \( \text{MAXDIVERSE}\text{SEt} \). Similarly, if we keep only the second test, we approximate \( \text{MAXdDISTANT} \). This method easily applies to diversity.

**Experiments**

We ran experiments using the Renault Megane Configuration benchmark (Amilhastre, Fargier, & Marguis 2002), a large real-world configuration problem. The variables represent various options such as engine type, interior options, etc. The problem consists of 101 variables, domain size varies from 1 to 43, and there are 113 table constraints, many of them non-binary. The number of solutions to the problem is over \( 1.4 \times 10^{12} \). We report the results obtained using the greedy algorithm and a Branch & Bound procedure taking advantage of the \( \text{Diverse}_C \) constraint. We solved \( \text{MAXDIVERSE}\text{Set} \) for \( k \) set to 3, motivated by work in recommender systems that argues that the optimal sized set to present to a user contains 3 elements (Shimazu 2001). The same variable ordering heuristics (\( \text{H}_{1, \text{dom}}/\text{deg} \times \text{i} \) in (Bessière, Chmeiss, & Saïs 2001)) was used for all instances. The values that were used the least in the previous solutions are chosen first. We also ran the same experiment without a specific value ordering. We also report results by simply shuffling the values in the domains and starting a new search without using a value ordering.

Figure 1: Solving \( \text{MAXDIVERSE}\text{SEt} \) for the Renault Megane configuration benchmark.

The results for the Renault benchmark are given in Figure 1. The distance between solutions, averaged over all instances, is plotted on the y-axis, against the cpu-time required to obtain it. To obtain different instances, upon which to base an average, we simply shuffled the value. For each method (i.e., greedy with value ordering, greedy without value ordering, and shuffling), the same initial solution is found, then the first curve corresponds to the first diverse solution from it, and the second curve to the next diverse solution (from both previous solutions).

We also ran some experiments on random binary CSPs, to observe how performance changes with the constrainedness of the problem. Results are given in Figure 2. All instances have 100 variables, domain size is 10, and 275 binary constraints. We generated 500 instances with tightness equal to 0.5 and 0.52 (0.52 is the phase transition), and 1500 instances with tightness equal to 0.53, as few of them were satisfiable.

The results on random CSPs show that our approximation method is more efficient as the constrainedness of the problem increases. For loosely constrained problems the simply shuffling method can be competitive since many possible diverse solutions exist. We also observe that choosing the value that is used the least in previous solutions tends to improve greatly performance on loosely constrained problems, but when the tightness of the constraints increases this value heuristic can slow the algorithm down as it chooses “successful” values last.

Overall, the optimization method without value ordering is more efficient on more tightly constrained problems. However, the same method with value ordering is consistently the best choice. Indeed, on easier problems the heuristic finds a distant solution quickly, whilst on harder problems the the Branch & Bound procedure combined with the \( \text{Diverse}_C \) constraint can find far more distant solutions than the shuffling approach.

The Renault Megane configuration problem is loosely constrained and thus admits a large number of solutions. It is not surprising therefore that the best method uses the value ordering heuristic. We are able to find 2 diverse solutions
In future work, we will study similarity and diversity problems that combine both similarity and diversity. Specifically, given a solution (or set of solutions), we wish to find a set of solutions that are as similar as possible to the given, but that are as mutually diverse as possible.

Acknowledgements. Hebrard and Walsh are supported by the National ICT Australia, which is funded through the Australian Government’s Backing Australia’s Ability initiative, in part through the Australian Research Council. Hnich is supported by Science Foundation Ireland (Grant 00/PI.1/C075). O’Sullivan is supported by Enterprise Ireland (Grant SC/02/289).

References

such that half of the variables are assigned differently within 3 seconds, or 3 such solutions within 15 seconds.

Related work
A number of researchers have studied the complexity of finding another solution to a problem. For example, given a graph and a Hamiltonian cycle with this graph, it is NP-complete to decide if the graph contains another Hamiltonian cycle. However, constraints are not typically placed on the relationship between the two solutions. One exception is work in constraint satisfaction where we ensure that the Hamming distance between the two solutions is maximized (Angelsmark & Thapper 2004; Crescenzi & Rossi 2002) or that the solutions are within some distance of each other (Bailleux & Marquis 1999). The problem classes introduced here subsume these problems.

Diversity and similarity are also important concepts in case-based reasoning and recommender systems (Bridge & Ferguson 2002; Smyth & McClave 2001; Shimazu 2001). Typically, we want chose a diverse set of cases from the case-base, but retrieve cases based on similarity. Also more recent work in decision support systems has focused on the use of diversity for finding diverse sets of good solutions as well as the optimum one (Løkketangen & Woodruff 2005).

Conclusions
Motivated by a number of real-world applications that require solutions that are either diverse or similar, we have proposed a range of similarity and diversity problems. We have presented a detailed analysis of their computational complexity (see Table 1), and developed a number of algorithms and global constraints for solving them. Our experimental results on some real-world problems are very encouraging.

In future work, we will study similarity and diversity problems that combine both similarity and diversity. Specifically, given a solution (or set of solutions), we wish to find a set of solutions that are as similar as possible to the given, but that are as mutually diverse as possible.

Acknowledgements. Hebrard and Walsh are supported by the National ICT Australia, which is funded through the Australian Government’s Backing Australia’s Ability initiative, in part through the Australian Research Council. Hnich is supported by Science Foundation Ireland (Grant 00/PI.1/C075). O’Sullivan is supported by Enterprise Ireland (Grant SC/02/289).

Table 1: A summary of the complexity results.

<table>
<thead>
<tr>
<th>Problem Class</th>
<th>Complexity Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>dDISTANCE-SET</td>
<td>NP-complete</td>
</tr>
<tr>
<td>dCLOSE-SET</td>
<td>NP-complete</td>
</tr>
<tr>
<td>MAXDIVERSE-SET</td>
<td>FPNP[log n]</td>
</tr>
<tr>
<td>MAXSIMILAR-SET</td>
<td>FPNP[log n]</td>
</tr>
<tr>
<td>MAXDISTANCE-SET</td>
<td>FPNP[log n]</td>
</tr>
<tr>
<td>MAXCLOSE-SET</td>
<td>FPNP[log n]</td>
</tr>
<tr>
<td>MOSTDISTANCE</td>
<td>FPNP[log n]</td>
</tr>
<tr>
<td>MOSTCLOSE</td>
<td>FPNP[log n]</td>
</tr>
</tbody>
</table>

**Table 1:** A summary of the complexity results. In future work, we will study similarity and diversity problems that combine both similarity and diversity. Specifically, given a solution (or set of solutions), we wish to find a set of solutions that are as similar as possible to the given, but that are as mutually diverse as possible.

**Acknowledgements.** Hebrard and Walsh are supported by the National ICT Australia, which is funded through the Australian Government’s Backing Australia’s Ability initiative, in part through the Australian Research Council. Hnich is supported by Science Foundation Ireland (Grant 00/PI.1/C075). O’Sullivan is supported by Enterprise Ireland (Grant SC/02/289).

**References**