Reformulating Constraint Satisfaction Problems to Improve Scalability

Kenneth M. Bayer[1], Martin Michalowski[2], Berthe Y. Choueiry[1,2], Craig A. Knoblock[2]

[1] Constraint Systems Laboratory, University of Nebraska-Lincoln
{kbayer,choueiry}@cse.unl.edu
[2] University of Southern California, Information Sciences Institute
{martinm,knoblock}@isi.edu

Abstract. Constraint Programming is a powerful approach for modeling and solving many combinatorial problems, scalability, however, remains an issue in practice. Abstraction and reformulation techniques are often sought to overcome the complexity barrier. In this paper we introduce four reformulation techniques that operate on the various components of a Constraint Satisfaction Problem (CSP) in order to reduce the cost of problem solving and facilitate scalability. Our reformulations modify one or more component of the CSP (i.e., the query, variables domains, constraints) and detect symmetrical solutions to avoid generating them. We describe each of these reformulations in the context of CSPs, then evaluate their performance and effects in on the building identification problem introduced by Michalowski and Knoblock [1].

1 Introduction

Choueiry et al. [2] proposed to characterize a reformulation as a transformation of a problem $\mathcal{P}$ from one encoding to another, where a problem is given by a *formulation* and a *query*, $\mathcal{P} = \langle \mathcal{F}, \mathcal{Q} \rangle$. The transformation may change the query and/or any of the components of the formulation. The goal of the transformation is to 'simplify' problem solving, where the benefit of the 'simplification' and other effects of the transformation are clearly articulated in the particular problem-solving context.

In this paper, we propose four reformulation techniques that operate on various aspects of a Constraint Satisfaction Problem (CSP) in order to improve the performance of problem solving. The problem formulation of a CSP is given by $\mathcal{F} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ where $\mathcal{V}$= $\{V_i\}$ is a set of variables, $\mathcal{D}$= $\{D_{V_i}\}$ the set of their respective domains, and $\mathcal{C}$ a set of constraints. A constraint is a relation over a subset of the variables specifying the allowable combinations of values for the variables in its scope. A solution is an assignment to the variables such that all constraints are satisfied. The query is usually to find one satisfying solution, in which case the problem is in **NP**-complete in general. Alternatively, the query could be to find all possible solutions.

In this paper, we describe a reformulation of a CSP as a transformation of the original problem $\mathcal{P}_\mathrm{o} = \langle \mathcal{F}_\mathrm{o}, \mathcal{Q}_\mathrm{o} \rangle$ into the reformulated problem $\mathcal{P}_\mathrm{r} = \langle \mathcal{F}_\mathrm{r}, \mathcal{Q}_\mathrm{r} \rangle$, where $\mathcal{F}_\mathrm{i}$ indicates a formulation and $\mathcal{Q}_\mathrm{i}$ indicates a query, as illustrated in Figure 1. While our reformulations apply to a variety of resource allocation problems, in this paper we describe their application to the building identification problem (BID) proposed by Michalowski and Knoblock [1]. The task is to assign a list of postal addresses to

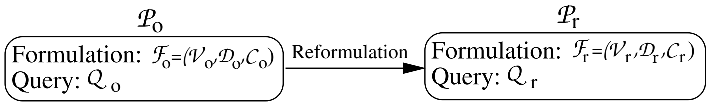

Fig. 1. The general pattern of a CSP transformation.

buildings appearing in a satellite image. A map provides the names of the streets and the positions of the buildings, but we do not know the addresses of the buildings or, for a building located on a street corner, on which street the building's address lies. The original work established the feasibility of modeling and solving this problem as a CSP. However, their work did not scale well to larger problems [1]. We show that the reformulations we introduce allow us to solve larger problems. The largest problem solved by Michalowski and Knoblock included 34 buildings. In this paper, we scale up to problems involving 206 buildings.

This paper is structured as follows. In Section 2 we introduce a new type of global constraint and describe how to use symbolic values to reformulate the domains of variables in the scope of this constraint. In Section 3 we present a reformulated query that reduces runtime in practice. In Section 4 we describe how relaxing an assignment problem into a matching problem can be used to improve the performance of backtrack search used to solve the assignment problem. We also describe a symmetry detection process for generating all solutions to a maximum matching in a bipartite graph from a single solution to the matching. In Section 5 we apply these techniques to the building identification problem (BID). We present experimental results that demonstrate their effectiveness in Section 6. Section 7 discusses related reformulation work and Section 8 contains our conclusions and future research directions.

2 Domain reformulation using symbolic values

We propose ALLDIFF-ATMOST as a global constraint useful for resource allocation problems. In this section we first describe this constraint. We then discuss how to reformulate the domains of the variables in the scope of this constraint in order to reduce their size both for general and totally ordered domains. Section 5.2 illustrates the use of this constraint and its reformulation in a practical resource allocation problem.

2.1 ALLDIFF-ATMOST

Example 1. An emerging country received an aid to build 7 hospitals on its territory, but does not want to put more than 2 hospitals in areas with high volcanic activity.

We propose the constraint ALLDIFF-ATMOST to model this situation. Given a set of variables $\mathcal{A} = \{V_1, V_2, ..., V_n\}$ with domains D_{V_i}, ALLDIFF-ATMOST$(\mathcal{A}, k, d)$, where $d \subseteq D_{V_i}$, $k \in \mathbb{N}$, and $k \leq |d|$, requires that (1) all variables be different and (2) at most k variables in $\mathcal{A}$ have values from d. Note that while the domains D_{V_i} may be different, d must be a subset of each one of them and D_{V_i}, and d and D_{V_i} may be finite or infinite.

Example 2. Consider with the variables $\mathcal{A}$=$\{V_1$, V_2, V_3, $V_4\}$ of a CSP, with D_i=$\{1$, 2, ..., 8$\}$ and the constraint ALLDIFF-ATMOST*($\mathcal{A}$, 2, $\{1, 3, 4, 5, 8\}$). The assignment $V_1 \leftarrow 5$, $V_2 \leftarrow 2$, $V_3 \leftarrow 7$, $V_4 \leftarrow 4$, and $V_5 \leftarrow 3$ satisfies the constraint.*

2.2 AllDiff-Atmost reformulation

Our first reformulation technique applies to the AllDiff-Atmost and the domains of the variables in its scope. The transformation, which we describe next, is theorem constant, in the sense that solutions to the reformulated problem map to solutions to the original problem [3]. The benefit of this reformulation is the reduction of the domain sizes. Because the complexity of many CP techniques depends on the sizes of the domains, the reformulation improves the solver performance.

We reformulate the domains of the variables in the scope of the constraint AllDiff-Atmost$(\mathcal{A}, k, d)$ by introducing k values s_l that we call *symbolic values* as follows:

$$\forall\, V_i \in \mathcal{A}\; D_{V_{i_\mathrm{r}}} = \{s_1, s_2, \ldots, s_k\} \cup (D_{V_i} \setminus d) \tag{1}$$

where the symbolic values s_j $(1 \leq j \leq k)$ can take any distinct values in d. Applying this reformulation on Example 2 yields the following domains for all four variables: D_{V_i}=$\{s_1$, s_2, 2, 6, 7$\}$, where s_1, s_2 can take any different values in $\{$1, 3, 4, 5, 8$\}$. In Example 1, the domains become $\{s_1$, $s_2\}$ $\cup$ $\{$sites in non-volcanic areas$\}$ where s_1, s_2 are different and range over sites with volcanic activities.

This reformulation procedure operates on the problem formulation and affects both the AllDiff-Atmost constraints and the domains of the variables in their scope, see Figure 2. However the most significant modification is the domain reformulation. We transform $\mathcal{D}_\mathrm{o}$ to $\mathcal{D}_\mathrm{r}$, where in $\mathcal{D}_\mathrm{r}$ the domains of variables in $\mathcal{A}$ are reformulated according to Equation (1). Replacing d in the original domains with k symbolic values reduces their sizes by $|d| - k$, which is useful when d is large or infinite.

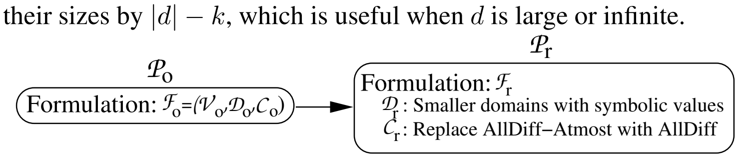

Fig. 2. The reformulation of AllDiff-Atmost.

This operation is particularly useful during backtrack search where the domain values are enumerated. If we want to assign 'ground' values to each symbolic value, we can do so as a post-processing step while ensuring that two symbolic values are always mapped back to distinct ground values. While a solution to the reformulated problem does not map to a unique solution to the original problem, we can generate any solution to the original problem from some solution to the reformulated problem.

Of particular concern is the interaction between this reformulation and the other constraints in the problem. When all the constraints in a problem can be checked on the symbolic values, as in the case of the BID, the reformulation is sound. When one or more constraints in a problem must be checked on the 'ground' values, then propagation must run on the appropriate representation for each constraint and, as soon as domain filtering causes $|d| \leq k$, then reformulated domains should be dropped and AllDiff-Atmost replaced with a AllDiff constraint.

2.3 Symbolic intervals

When the values in the variables domains follow a total order, as in numeric domains, the domains are commonly represented as intervals and constraint propagation is typically restricted to the endpoints of these intervals, as in box-consistency algorithms. The reformulation of an ALLDIFF-ATMOST in the presence of totally ordered domains obviously remains valid. However, in order to *restrict propagation to the endpoints of the intervals* representing the domains, the following is needed:

1. We require the values in d to form a convex interval.
2. We must add ordering constraints between two consecutive s_i: $s_1 < s_2 < \ldots < s_k$.
3. We must add total ordering constraints between the two extreme symbolic values, s_1 and s_k, and their closest neighbors in the reformulated domains, which is accomplished as follows. Let $D^{\mathrm{l}}_{V_{i\,r}}$ and $D^{\mathrm{r}}_{V_{i\,r}}$ be respectively the intervals of $D_{V_i} \backslash d$ to the left and right of, and adjacent to, d. The right endpoint of $D^{\mathrm{l}}_{V_{i\,r}}$ must be less than s_1, and the left endpoint of $D^{\mathrm{r}}_{V_{i\,r}}$ must be greater than s_k. See Figure 3.

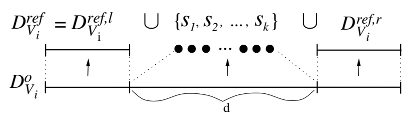

Fig. 3. ALLDIFF-ATMOST reformulation for totally ordered domains.

4. When mapping the symbolic values back to ground values, the ground values must respect the total ordering imposed on the symbolic values.

Section 5.2 illustrates the use of ALLDIFF-ATMOST and its reformulation on the BID.

3 Query Reformulation

In a CSP, the query is usually to find a single solution (satisfiability problem) or all solutions (enumeration problem). However, in some applications, we may not be interested in entire solutions, but rather all the values that each variable takes in any solution. We call the problem corresponding to this query a *per-variable solutions*.[3] Thus, we reformulate the query from $\mathcal{Q}_{\mathrm{o}}$, enumerating all solutions, to $\mathcal{Q}_{\mathrm{r}}$, finding a per-variable solution. Formally, we define $\mathcal{Q}_{\mathrm{r}}$ as $\forall V_i$, $x \in D_{V_i}$, find if $\mathcal{P}_{\mathrm{o}} \wedge (V_i \leftarrow x)$ is satisfiable. Figure 4 illustrates this reformulation. This reformulation changes the problem, because the solution returned will be different. However, in some cases, the per-variable solution is an acceptable alternative. This transformation changes the complexity class of the problem from a counting problem to a satisfiability one.

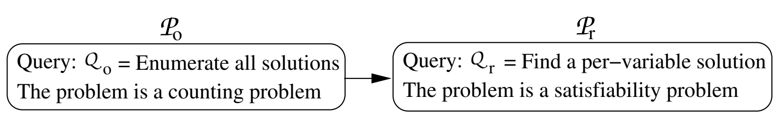

Fig. 4. Query reformulation.

In this section we propose an algorithm to find per-variable solutions, and describe how this algorithm can be used to enforce high levels of relational consistency.

[3] Formally, this query corresponds to finding the minimal CSP.

3.1 Per-variable solutions

We can find the set of possible assignments to each variable in a CSP by solving the enumeration problem, that is finding all solutions and then iterating through the solution set to collect the values taken by each variable. However, the number of solutions to a CSP is $O(d^n)$, where n is the number of variables and d is the maximum domain size. Thus, finding all solutions may be prohibitively expensive.

We replace the query of finding all solutions (an enumeration problem) with the query of finding if a solution exists for every combination of variable-value pair (a polynomial number of satisfiability problems). Algorithm 1 tests for every variable-value pair (V_i, x) if the CSP with $V_i \leftarrow x$ is solvable. When a solution exist, x is added to the data structure returned by the algorithm. The algorithm returns the set of variables along with all their values that appear in a solution.

Input: $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$
Output: S, a per-variable solution
foreach $V_i \in \mathcal{V}$ **do**
    $S[V_i] \leftarrow \emptyset$
end
foreach $V_i \in \mathcal{V}$ **do**
    foreach $x \in D_{V_i}$ **do**
        if $\mathcal{P}$ *with* $V_i \leftarrow x$ *has a solution* **then**
            $S[V_i] \leftarrow S[V_i] \cup \{x\}$
        end
    end
    if $|S[v]| = 0$ **then**
        return $\mathcal{P}$ *has no solutions*
    end
end
return S

Algorithm 1: Finding the per-variable solutions.

The inner loop of the algorithm runs $O(nd)$ times. Each iteration requires determining the satisfiability of a CSP. This operation appears costly, but in cases where the original CSP has significantly more than nd solutions, Algorithm 1 can perform significantly better than enumerating all solutions to the CSP.

3.2 Relational consistency

When solving a CSP, it is often beneficial to make the constraint network arc-consistent. Enforcing arc-consistency filters values from the variable domains that cannot exist in any solution to the problem. We can perform even more filtering by considering higher levels of consistency. Dechter and van Beek introduced *relational* (i, m)*-consistency* as the consistency of m non-binary constraints over every subset of i variables in the CSP [4]. Dechter [5] proposed the algorithm $RC_{(i,m)}$ for computing relational (i, m)-consistency. $RC_{(i,m)}$ works as follows. For every set C_m of m constraints in a constraint network, compute the join of the m constraints and project the result onto each subset of i variables. The algorithm is not practical for high values of m, because the memory requirements for computing and storing a join of m constraints rises exponentially with the number of variables.

Algorithm 1 computes a minimal network, which means that every value remaining in the network appears in at least one solution. Thus, the resulting network is the same as if we had executed $RC_{(1,m)}$. The difference between the two algorithms is that Algorithm 1 is polynomial space, whereas $RC_{(1,m)}$ is exponential space. We could generalize Algorithm 1 to consider sets of up to i variables rather than unique ones. This extension would allow the algorithm to produce the same results as $RC_{(i,m)}$, where the memory requirement rises with i, which quickly becomes impractical.

4 Constraint relaxation for problem reformulation

At the core of many resource allocation problems lies the problem of matching between the elements of two sets: the tasks and the resources. In general, the problem may be complex (and likely intractable). However, in some cases, we may be able to identify constraints that can be removed to reduce the original problem into a matching problem in a bipartite graph.

Removing (or adding) a constraint in a problem formulation to yield a necessary (or sufficient) tractable approximation of the problem is a typical reformulation strategy. Examples abound and include: In AI, admissible heuristics generation for A* [6] and theory approximation [7]; in mathematical programming, linear relaxation of integer programs, Lagrangian relaxation [8], and the cutting-plane method.

In this section we first describe the relaxation of a resource allocation problem into a matching problem in a bipartite graph. Then, we describe techniques that take advantage of this reformulation when performing search for solving a CSP. Finally, we describe a symmetry detection technique that allows us to generates all the possible matchings in a bipartite graph from a single matching.

4.1 Matching as a relaxation

Let $G = (X \cup Y, E)$ be a bipartite graph with edge set E, vertex set $V = X \cup Y$, and partitions X and Y, which are independent sets of vertices. We define a *match count* for each vertex in $v \in V$, which we denote $m(v)$, to be a positive (non-null) integer. A *matching* in G is a set of edges $M \subseteq E$ such that $\forall v \in V$ there exists at most one edge $e \in M$ incident to v. In this paper we consider a matching in G to be a set of edges $M \subseteq E$ such that $\forall v \in V$ there exists at most $m(v)$ edges $e \in M$ incident to v. Further, we say that a matching M saturates vertex v iff M has exactly $m(v)$ edges incident to v; and a matching M saturates a set S iff M saturates all vertices in S. Finding a matching that saturates S can be done in polynomial time (see Section 5.5).

We propose to reformulate a resource allocation problem by relaxing it into a matching problem in a bipartite graph that saturates one of the graph's two partitions. Figure 5 illustrates this relaxation. While the original problem may be intractable, the reformulated one can be efficiently solved (i.e., in polynomial time). When the reformulated problem is not solvable, the more constrained original problem is not solvable. However, the solvability of the reformulated problem does not guarantee that of the original problem. Our reformulation is thus a necessary approximation [9].

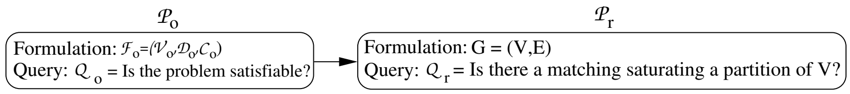

Fig. 5. Relaxation of a CSP as a matching problem.

4.2 Integrating the matching relaxation in backtrack search

When modeling a resource allocation problem as a CSP and solving it with backtrack search, we can take advantage of the relaxed problem in two ways:

1. As a *preprocessing step* prior to search, and
2. As *a lookahead mechanism* during search to filter out, from the domains of the future variables, those values that cannot yield a solution.

Prior to search, if we determine that the relaxed problem is not soluble, we can safely avoid using search. Further, during search, we can adapt the algorithm of Régin [10], which finds all edges of the bipartite graph that do not participate in *any* covering matching, to identify, in one step, all values in the domains of all future variables that do not participate in any saturating matching. This single operation allows us to filter the domains of all future variables in one step.

The reformulation into a matching problem is especially useful when finding per-variable solutions, because Algorithm 1 executes nd satisfiability tests. We propose to test, after line 5 in Algorithm 1, whether the relaxed problem is solvable, and proceed to line 6 only if this test succeeds. Otherwise, we return to line 5.

4.3 Generating solutions by symmetry

The set of maximum matchings in a bipartite graph can be obtained by enumerating all maximum matchings using an algorithm such as the one proposed by Uno [11]. In this section, we characterize all maximum matchings in a bipartite graph as symmetric to a single base matching, and proposed to use this symmetry to enumerate all solutions.

Our symmetry detection relies on two graph constructions described by Berge [12]: *alternating cycles* (AltCyc) and *even alternating paths starting at a free vertex* (EvAltP). An AltCyc or EvAltP in a graph G relative to a matching M alternate between edges in M and edges not in M. If we take a maximum matching M and a AltCyc or EvAltP P, we can produce another maximum matching M' by computing the symmetric difference of M and P, denoted $M \Delta P$. We use that mechanism to identify all maximum matchings in a bipartite graph G as symmetric of a single maximum matching M. Let $\mathcal{S}$ be the set of all AltCyc's and EvAltP's relative to M. We construct another maximum matching M_i by choosing a disjoint subset $\mathcal{S}_i \subseteq \mathcal{S}$ and computing $M \Delta \mathcal{S}_i$. M_i is symmetrical to M in that it is identical to M in all edges except those in $\mathcal{S}_i$. In fact, for any maximum matching M_j of G, we prove that there exists an $\mathcal{S}_j$ such that $M_j = M \Delta \mathcal{S}_j$ using Lemma 3.1.9 of [13]. We generate $\mathcal{S}$ by first orienting G using the construction described by Hopcroft and Karp [14]. From the oriented graph, we enumerate the alternating paths by finding all EvAltP's, as defined by Berge [12]. We enumerate the AltCyc's from the strongly connected components in the oriented graph as described by Régin [10]. Thus, to store the information necessary to enumerate all alternating paths and cycles, and therefore all maximum matchings, we only need to store a single base matching, the set of free vertices, and the set of strongly connected components[4].

Consider the bipartite graph $G = (X \cup Y, E)$, where X = $\{x_1, x_2, x_3, x_4\}$, Y = $\{y_1, y_2, y_3\}$, and E=$\{(x_1, y_1), (x_2, y_1), (x_2, y_2), (x_3, y_2), (x_3, y_3), (x_4, y_2), (x_4, y_3)\}$. Figure 6 (a) shows a maximum matching M in G. $P = x_1y_1x_2$ is an alternating path and $C = x_3y_2x_4y_3x_3$ is an alternating cycle. We find other maximum matchings using the symmetric difference operator. Figure 6 (b) show $M \varDelta P$, Figure 6 (c) shows $M \varDelta C$, and Figure 6 (d) shows $M \varDelta (C \cup P)$.

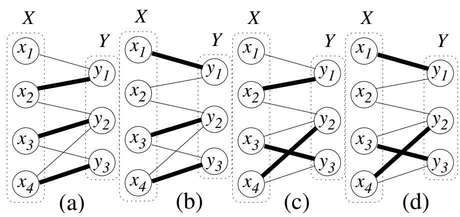

Fig. 6. Multiple matchings saturating Y.

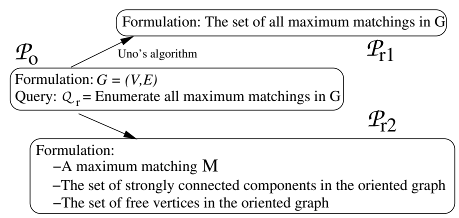

Fig. 7. Finding all maximum matchings.

Figure 7 illustrates the two reformulations of $\mathcal{P}_o$, the problem of enumerating all maximum matchings. We can reformulate $\mathcal{P}_o$ as $\mathcal{P}_{r1}$, the set of all maximum matchings, using Uno's algorithm. Alternatively, we can reformulate the problem as $\mathcal{P}_{r2}$, a base matching and its corresponding sets of strongly connected components and free vertices. All matchings can be enumerated from $\mathcal{P}_{r2}$ as needed. Our construction has the same time complexity as Uno's, which is linear in the number of maximum matching. However, our characterization of the solutions as symmetries is a valuable one:

1. It provides a more compact representation of the set of solutions. Rather than storing all matchings, we store a single matching, a set of strongly connected components, and a set of free vertices.
2. In case one is indeed seeking *all*, or a given number of, the solutions to BID (similarly, to a resource allocation problem that has a maximum matching relaxation), we can generate every symmetric to that known single matching and test if it satisfies the additional constraints of the non-relaxed problem, when it does not, the matching is a solution to the non-relaxed problem found without search. Naturally, the number of maximum matchings can be large.

We do not currently exploit those features, but they deserve further investigations.

5 Application to the building identification problem

We apply the four techniques presented above to the BID [1]. The task is to assign a list of addresses from a phone book to buildings appearing in a satellite image. Each address consists of the combination of a street name and a number. A map provides the

[4] An improvement suggested by a anonymous reviewer.

names of the streets and the positions of the buildings. The map could come from an online source or a satellite image. We know the street names and the positions of the buildings, but we do not know the addresses of the buildings or, for a building located on a street corner, on which street the building's address lies. A variety of data sources, such as a phone books, gazetteers, or property records, provide at least a partial list of addresses in a region. We generically refer to the addresses given as input as phone-book addresses regardless of their actual source. Figure 8 shows a BID instance.

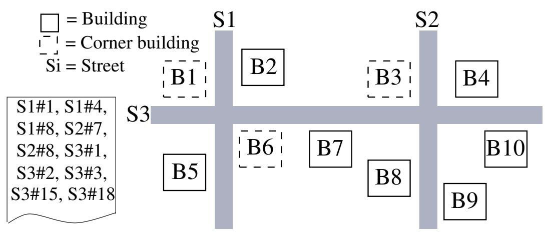

Fig. 8. An example of the BID problem.

In general, the phone book may be incomplete, listing fewer addresses than there are buildings in the image, but the reverse does not hold. We must map every phone-book address to a building in the image. Michalowski and Knoblock established the feasibility of modeling and solving this problem as a CSP. However, their work did not scale well to larger problems. We show in Section 6 that the reformulations we propose allow us to solve larger problems.

5.1 CSP model

Below we describe the variables and constraints in our CSP model of the BID, which improves on the one proposed in [1].

Our model uses three types of variables: *orientation* variables, *corner* variables, and *building* variables. In general, there are four Boolean *orientation variables*: Increasing-North, IncreasingEast, OddOnEastSide, and OddOnNorthSide. The first two are *ordering* variables and indicate whether or not addresses increase in value when moving toward the north and to the east. The remaining two are *parity* variables and indicate on which side of the street odd addresses occur. The *corner variables* represent the possible *streets* on which a corner building might be. We generate one corner variable for each corner building, whose domain is the list of streets on which the building could lie, and has size 2 in most cases. The corner buildings are natural 'articulations' in the constraint network: once the solver assigns values to all corner buildings, the constraint network degenerates into a set of chains (corresponding to buildings along street segments) that can be solved in a backtrack-free manner. Michalowski and Knoblock too noted this feature [1]. Thus, the solver instantiates corner buildings as soon as possible. The *building variables* represent the addresses (i.e., numbers) of the buildings. We generate a building variable for every building on the map. The domain of a variable is every possible address on the building's streets.

Our model has five types of constraints: *parity*, *ordering*, *corner*, *phone book*, and *grid*. *Parity constraints* are binary constraints and ensure that the numbers assigned to buildings respect the values assigned to the parity (orientation) variables. *Ordering constraints* are ternary constraints, and link an ordering variable to two building variables along the same street. These constraints ensure that the addresses assigned to the building variables respect the ordering specified by the ordering variable. *Corner constraints* link the the corner and building variables of a corner building and ensure that the address assigned to the building is consistent with the street chosen for the building. *Phone-book constraints* exist for each street on the map. These constraints ensure that the solver assigns every address in the phone book to some building along that street. These constraints usually have a high arity, because their scope is the set of buildings along the street. *Grid constraints* exist between buildings across certain artificial grid-lines, depending on the region we are modeling. These constraints ensure that the addresses of adjacent buildings across the grid-lines are in separate numeric increments. For example, in many cities in the United States, addresses increase to the next increment of 100 across intersections.

5.2 Symbolic values

If the phone book is incomplete, we must infer the missing numbers to add to the variables' domains. Michalowski and Knoblock proposed to enumerate all numbers between 1 and the largest address that appears on the street [1]. Their approach has two problems. First, the choice of the upper limit is arbitrary. When the largest address is not in the phone book, this approach may yield incorrect solutions. The second problem with this approach is that the size of the domains becomes prohibitively large on real-world data. Using symbolic values in phone-book constraint solves both problems.

Let S be a street, P_S its set of phone-book addresses of a given parity, B_S the set of buildings on the side of S of that parity, and [min,max] the range of address numbers on that side of S. The address numbers in P_S partition [min,max] into consecutive convex intervals. In any such interval (i_1, i_2), we cannot use more than $|B_S|-|P_S|$ addresses, which we express as ALLDIFF-ATMOST(B_S, k_a, (i_1, i_2)) with k_a=minimum($|B_S|$-$|P_S|$, $\lfloor\frac{(i_2-i_1)-1}{2}\rfloor$). We reduce the variables' domains using the reformulation of Section 2 on each of these intervals. For example, assume we have, on the even side of S, B_S=$\{B_1, B_2, \ldots, B_5\}$, P_S=$\{$S#12,S#18$\}$, min=2, and max=624. An assignment cannot use more than 3 numbers in each of [2,12), ([12,18), and (18,624], yielding 3 ALLDIFF-ATMOST constraints with the following arguments (B_S, 3, [2,12)), (B_S, 2, (12,18)), and (B_S, 3, (18,624]). We replace the domain [1,624] of each variable with the significantly smaller set $\{s_1, s_2, s_3, 12, s_4, s_5, 18, s_6, s_7, s_8\}$ where s_1, s_2, $s_3 \in$ [2,12), s_4, $s_5 \in$ (12,18), and s_6, s_7, $s_8 \in$ (18,624] and $s_i < s_j$ for $i < j$, see Figure 9. This process allows us to choose an arbitrarily large upper bound (max) on a given street.

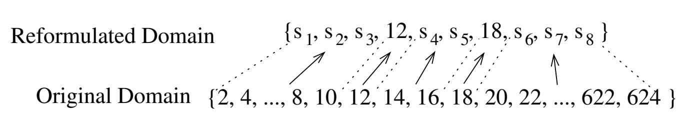

Fig. 9. Domain reformulation for the BID.

5.3 Query reformulation

Another challenge of real-world BID instances is the large number of solutions. If the phone book is incomplete, the problem is under-constrained, yielding a large number of solutions. One possible query would be to enumerate all the solutions to collect the acceptable list of addresses for each building [1]. By reformulating the query as proposed in Section 3, we can use Algorithm 1 to obtain the same result at a much cheaper cost. In summary, we replace the query: "Enumerate all solutions and collect the addresses taken by the buildings in these solutions" with the query "Find all the addresses that a given building can take."

5.4 Constraint relaxation for problem reformulation

We show below that, when no grid constraints exist, the BID problem can be modeled as a matching in a bipartite graph, and is thus tractable. The CSP approach of Michalowski and Knoblock remains pertinent in that it allows one to represent arbitrary street-addressing schemas used around the world, such as grid constraints. We propose the removal of grid constraints as a tractable relaxation of the BID.

Given an instance of this problem without grid constraints, we construct a bipartite graph $G = (B \cup S, E)$ as follows. First, assume an assignment to the orientation variables (there are 2^4 such assignments). For each building β in the problem, add a vertex b to B. For each street σ in the problem, add two vertices s_{odd} and s_{even} to S, one for each side of the street. For each building β, add an edge between vertex b and the street vertex corresponding to the street side on which β may be. (Note that corner buildings are on two streets.) Assuming odd numbers appear on the North and West sides of the street, Figure 10 shows the construction of G for the map in Figure 8. We can show that a matching in this graph that saturates S corresponds to a satisfactory assignment of streets to corner buildings.

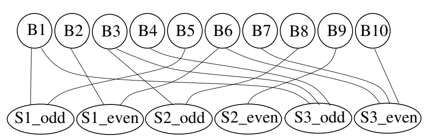

Fig. 10. Graph construction for Figure 8.

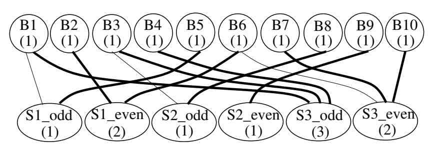

Fig. 11. Satisfying matching for Figure 10.

Figure 11 shows a satisfying matching for the graph from Figure 10, where the edges included in the matching are darkened. The numbers in parentheses indicate the match count of the vertex. This matching corresponds to an assignment of buildings to streets. Thus, we can now construct a solution to the problem as a post-processing step, where we simply assign numbers to each building along each street.

While the matching approach is powerful, it cannot model the grid constraint. Whether the problem with the grid constraints can be solved efficiently remains an open question.

We propose to use the matching in 2 ways: (1) directly solve problems that have no grid constraints, and (2) use the relaxation to detect unsolvability (both as a preprocessing step and for lookahead as discussed in Section 4).

5.5 Solvers

It was simply impossible to solve any of our real-world data sets using the original query. We could solve them only with the reformulated query, using Algorithm 1.

We implemented two solvers: a matching-based solver and a search-based solver. The former finds a maximum matching using an $O(n^{5/2})$ algorithm by Hopcroft and Karp [14] after replacing each vertex in the bipartite graph by as many vertices as its match count. The latter uses backtrack search (BT) with nFC3, a look-ahead strategy for non-binary CSPs [15], and conflict-directed backjumping [16]. In BT, we implemented a hybrid representation of the domains: enumerated values and intervals. We use the interval representation to propagate ordering constraints (i.e., less-than constraints), and restrict this propagation to the bounds of the intervals without loss of pruning power. This representation improved our runtime by one order of magnitude.

When the problem given as input has no grid constraint, we use the matching solver in line 6 of Algorithm 1, which computes one matching in each of the nd loops. Thus, the solver runs in polynomial time, which is a significant improvement compared with the exponential-time backtrack search based solver. When the problem instance has grid constraints, we proceed as follows:

1. Preprocessing: We insert a call to the matching solver after line 5 in Algorithm 1 and proceed to line 6 if we find a matching, otherwise return to line 5.
2. Backtrack search (BT): We use the search-based solver in line 6.
3. Lookahead: In addition to nFC3, we filter in one step the domains of all future variables given the current path in the search (see Section 4).
   Figure 12 illustrates the behavior of the solvers.

6 Experiments

Table 1 describes the properties of the regions of the city of El Segundo (CA), on which we ran our experiments. The number of calls refers to the total number of times to line 6 of Algorithm 1. Each call to line 6 was timed out after one hour. We report the number of timed out executions.

The completeness of the phone book indicates what percent of the buildings on the map have a corresponding address in the phone book. We created the complete phone books using property-tax data, and the incomplete phone books using the real-world phone-book.

Effect of domain reformulation. Table 2 shows the benefit of domain reformulation by comparing the performance when using the original domains or the reformulated domains. The experiment uses backtrack search (BT), but does not take into consideration the grid constraints. When the phone book is complete, no ALLDIFF-ATMOST constraints are present, and thus the reformulation does nothing. The advantage of the reformulation is clear when using the incomplete phone book.

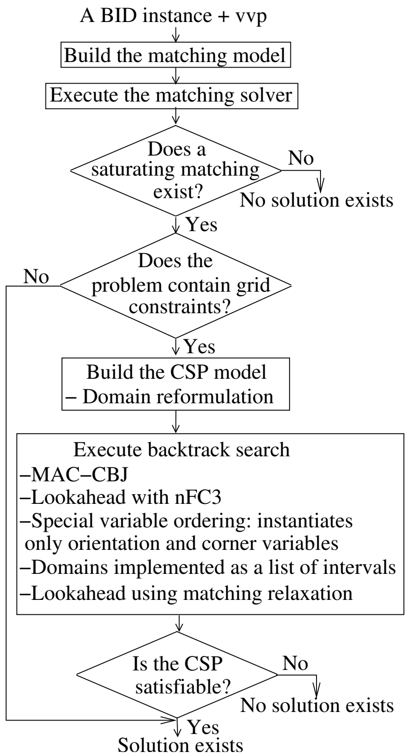

Fig. 12. Implementing Line 6 of Algorithm 1.

Table 1. Case studies used in experiments.

Case study	Phone book completeness	Number of bldgs	crnr bldgs	blks	calls
NSeg125-c	100.0%	125	17	4	4160
NSeg125-i	45.6%				1857
NSeg206-c	100.0%	206	28	7	4879
NSeg206-i	50.5%				10009
SSeg131-c	100.0%	131	36	8	3833
SSeg131-i	60.3%				2375
SSeg178-c	100.0%	178	46	12	4852
SSeg178-i	65.6%				2477

Table 2. Effect of domain reformulation.

Case study	Avg. domain size		Runtime [sec]		Timeouts	
	Orig.	Ref.	Orig.	Ref.	Orig.	Ref.
NSeg125-i	1103.1	236.1	2943.7	744.7	0	0
NSeg206-i	1102.0	438.8	14818.9	5533.8	0	0
SSeg131-i	792.9	192.9	67910.1	66901.1	18	17
SSeg178-i	785.5	186.3	119002.4	117826.7	32	29

Table 3. Solvers' performance (no grid).

Case study	Runtime [sec] BT	Matching	Matching + Symmetry
NSeg125-c	139.2	4.8	0.03
NSeg125-i	744.7	2.5	*
NSeg206-c	4971.2	16.3	0.06
NSeg206-i	5533.8	8.5	*
SSeg131-c	38618.3	7.3	0.26
SSeg131-i	66901.1	3.1	*
SSeg178-c	117279.1	22.5	0.41
SSeg178-i	117826.7	4.9	*

* Did not finish in 1 hour.

Effect of query reformulation. As stated in Section 5.5, the sheer number of solutions made it impossible to solve problem instances with incomplete phone-books using the query of enumerating all solutions. Thus, without the query reformulation, we would not have been able to solve the incomplete phone-book instances.

Effect of finding symmetrical maximum matchings. In the absence of grid constraints, the BID can be solved in polynomial time by the matching solver. Here we compare backtrack search, a solver that uses Algorithm 1 with a matching solver, and a solver that uses the reformulation of symmetric matchings from section 4.3. Finding all symmetric matchings requires enumerating all matchings, which isn't feasible for the under-constrained incomplete phone-book problems. Thus, those problem instances timed out and are indicated by asterisks. However, when the number of solutions was small, such as when the phone-book is complete, the symmetry solver had significantly better performance than the per-variable matching solver. The benefit in terms of runtime reduction is shown in Table 3.

Effect of relaxing a CSP into a matching problem. To test the use of the matching relaxation as a preprocessing step and lookahead mechanism, we added grid constraints to each region. Table 4 shows the results of these experiments, comparing the performance of: (1) the backtrack search (BT), (2) BT with matching for preprocessing (Preproc+BT), (3) BT with matching for lookahead (Lkhd+BT), and (4) BT with matching for both purposes (Preproc+BT+Lkhd). We report runtime, number of timeouts, and number of calls to the CSP solver saved by the preprocessing. In all cases, the same solutions were found. Our results indicate that, in general, the integration of the matching and BT improves performance. There are exceptions, when the cost of the additional processing exceeds the gains in terms of reduced search space. However, even when we saw performance degradation, the degradation was minimal.

Table 4. Improvements due to preprocessing and lookahead.

NSeg125-c + grid	CPU [sec]	#Timeouts	Calls saved
BT	100.8	0	-
Preprocessing+BT	33.2	0	97.0%
BT+Lkhd	140.2	0	-
Preproc+BT+Lkhd	39.6	0	97.0%
NSeg125-i + grid	**CPU [sec]**	**#Timeouts**	**Calls saved**
BT	1232.5	0	-
Preprocessing+BT	1159.1	0	62.6%
BT+Lkhd	726.6	0	-
Preproc+BT+Lkhd	701.1	0	62.6%
NSeg206-c + grid	**CPU [sec]**	**#Timeouts**	**Calls saved**
BT	2277.5	0	-
Preprocessing+BT	614.2	0	98.9%
BT+Lkhd	1559.2	0	-
Preproc+BT+Lkhd	443.8	0	98.9%
NSeg206-i + grid	**CPU [sec]**	**#Timeouts**	**Calls saved**
BT	4052.8	0	-
Preprocessing+BT	3806.7	0	87.8%
BT+Lkhd	3499.5	0	-
Preproc+BT+Lkhd	3510.0	0	87.8%

SSeg131-c + grid	CPU [sec]	#Timeouts	Calls saved
BT	17063.3	0	-
Preprocessing+BT	5997.9	0	92.5%
BT+Lkhd	9745.8	0	-
Preproc+BT+Lkhd	4256.0	0	92.5%
SSeg131-i + grid	**CPU [sec]**	**#Timeouts**	**Calls saved**
BT	114405.9	30	-
Preprocessing+BT	114141.3	29	74.2%
BT+Lkhd	107896.3	30	-
Preproc+BT+Lkhd	108646.5	30	74.2%
SSeg178-c + grid	**CPU [sec]**	**#Timeouts**	**Calls saved**
BT	78528.6	14	-
Preprocessing+BT	15717.9	1	91.9%
BT+Lkhd	74172.0	14	-
Preproc+BT+Lkhd	13961.1	1	91.9%
SSeg178-i + grid	**CPU [sec]**	**#Timeouts**	**Calls saved**
BT	138404.2	35	-
Preprocessing+BT	103244.7	25	72.7%
BT+Lkhd	121492.4	32	-
Preproc+BT+Lkhd	85185.9	22	72.7%

7 Related work

Reformulation has been applied to a wide range of CSP problems with much success. The literature encompasses also approaches to modeling, abstraction, approximation, and symmetry detection. Nadel studied 8 different models of the n-Queens problem, some of which much easier to solve than others [17]. Glaisher proposed avoiding symmetry in the Eight Queens as far back as 1874 [18]. This topic has recently received increased attention, for example in the work of Puget [19] and Ellman [9]. Holte and Choueiry provide a general discussion on abstraction and reformulation in AI including CSPs [20]. Razgon et al. [21] introduced a class of problems they called Two Families of Sets constraints (TFOS), and a technique for reformulating TFOS problems into network flow problems. Conceptually, the relaxed problem we study in Section 4 constitutes a special case of the TFOS problem.

8 Conclusions and future work

We introduced four general reformulation techniques for CSP, and integrated them in a a comprehensive framework for solving the BID while highlighting their usefulness for general CSPs. For example, our query reformulation facilitates a much wider use of relational consistency algorithms than was possible before. In the future, we intend to evaluate these techniques in other application settings. For example, we believe that many resource allocation problems have matching relaxations like we described.

Acknowledgments. Experiments were conducted on the Research Computing Facility at UNL. This research is supported by NSF CAREER Award #0133568 and the Air Force Office of Scientific Research under grant numbers FA9550-04-1-0105 and FA9550-07-1-0416. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of any of the above organizations or any person connected with them.

References

1. Michalowski, M., Knoblock, C.: A Constraint Satisfaction Approach to Geospatial Reasoning. In: Proc. of AAAI 2005. (2005) 423–429
2. Choueiry, B.Y., Iwasaki, Y., McIlraith, S.: Towards a Practical Theory of Reformulation for Reasoning About Physical Systems. Artificial Intelligence **162 (1–2)** (2005) 145–204
3. Giunchiglia, F., Walsh, T.: A Theory of Abstraction. Artificial Intelligence **57**(2-3) (1992) 323–389
4. Dechter, R., van Beek, P.: Local and global relational consistency. Journal of Theoretical Computer Science (1996)
5. Dechter, R.: Constraint Processing. Morgan Kaufmann (2003)
6. Russell, S., Norvig, P.: Informed Search and Exploration, page 107. In: Artificial Intelligence: A Modern Approach. Prentice Hall (2003)
7. Selman, B., Kautz, H.: Knowledge Compilation and Theory Approximation. Journal of the ACM **43**(2) (1996) 193–224
8. Milano, M., ed.: Constraint and Integer Programming: Toward a Unified Methodology. Kluwer (2004)
9. Ellman, T.: Abstraction via Approximate Symmetry. In: IJCAI 93. (1993) 916–921
10. Régin, J.: A filtering algorithm for constraints of difference in csps. In: AAAI 1994. (1994) 362–367
11. Uno, T.: Algorithms for Enumerating All Perfect, Maximum and Maximal Matchings in Bipartite Graphs. In: Int. Symp. on Algorithms and Comput. (ISAAC '97). (1997) 92–101
12. Berge, C.: Graphs and Hypergraphs. American Elsevier (1973)
13. West, D.: Introduction to Graph Theory. 2nd edn. Prentice Hall (2001)
14. Hopcroft, J., Karp, R.: An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. SIAM **2** (1973) 225–231
15. Bessière, C., Meseguer, P., Freuder, E., Larrosa, J.: On Forward Checking for Non-binary Constraint Satisfaction. In: CP 99. (1999) 88–102
16. Prosser, P.: MAC-CBJ: Maintaining Arc Consistency with Conflict-Directed Backjumping. Technical Report 95/177, Univ. of Strathclyde (1995)
17. Nadel, B.: Representation Selection for Constraint Satisfaction: A Case Study Using n-Queens. IEEE Expert **5**(3) (1990) 16–24
18. Glaisher, J.: On the Problem of the Eight Queens. Philosophical Magazine **4**(48) (1874) 457–467
19. Puget, J.: On the satisfiability of symmetrical constraint satisfaction problems. In: ISMIS93. (1993) 350–361
20. Holte, R.C., Choueiry, B.Y.: Abstraction and Reformulation in Artificial Intelligence. Philosophical Trans. of the Royal Society Sec. Biological Sciences **358**(1435) (2003) 1197–1204
21. Razgon, I., O'Sullivan, B., Provan, G.: Generalizing Global Constraints Based on Network Flows. In: Workshop on Constraint Modelling and Reformulation. (2006) 74–87