Reformulating CSPs for Scalability with Application to Geospatial Reasoning

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. While many real-world combinatorial problems can be advantageously modeled and solved using Constraint Programming, scalability remains a major issue in practice. Constraint models that accurately reflect the inherent structure of a problem, solvers that exploit the properties of this structure, and reformulation techniques that modify the problem encoding to reduce the cost of problem solving are typically used to overcome the complexity barrier. In this paper, we investigate such approaches in a geospatial reasoning task, the building-identification problem (BID), introduced and modeled as a Constraint Satisfaction Problem by Michalowski and Knoblock [1]. We introduce an improved constraint model, a custom solver for this problem, and a number of reformulation techniques that modify various aspects of the problem encoding to improve scalability. We show how interleaving these reformulations with the various stages of the solver allows us to solve much larger BID problems than was previously possible. Importantly, we describe the usefulness of our reformulations techniques for general Constraint Satisfaction Problems, beyond the BID application.

1 Introduction

Geospatial data integration aims at combining geospatial information from traditional and non-traditional data sources to infer information that is not available in any one source. The inadvertent bombing of the Chinese Embassy in Belgrade [2] illustrates the importance of geospatial data integration. That event could have been avoided by reasoning about the information that was available at the time (i.e., telephone books and maps) to identify the buildings shown in a satellite image. More generally, the information gained by data integration can be used to verify and augment geospatial databases (e.g., gazetteers), and extend the capabilities of geospatial systems (e.g., Google Maps, Google Earth, and Microsoft VirtualEarth).

Michalowski and Knoblock [1] identified and studied the Building Identification (BID) problem as an application of significant intelligence and civilian impact. The task is to assign a potentially incomplete list of postal addresses, collected from various 'phone-book' sources, to 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. They modeled the problem as a Constraint Satisfaction Problem (CSP) and used an existing solver (CPlan [3]) to find *all possible* matchings of addresses to buildings that are consistent with the phone book and with the geographical layout in the image. Their work established the feasibility of the approach and *identified an important new area where CP techniques are useful for solving real-world problems*. However, their approach resisted scaling because their model included high-arity constraints and their generic solver failed to take advantage of the structural information in the application domain. While we show in this paper that the particular BID problem studied in [1] is tractable, it is clear that only a careful theoretical study can determine whether or not a given set of constraints in the BID problem yields a tractable problem. The value of a CP approach is its flexibility in solving new problems with arbitrary constraints even when the problem's tractability is unknown. This paper addresses the scalability of the CP approach to the BID problem with the use of reformulation techniques, and discusses the use of the proposed reformulations to general CSPs.

First, we propose an improved constraint model that reflects the topology of the streets layout, and accommodates the addition of new constraints locally to express variations of street-numbering schemas around the world. *Second*, we introduce a custom solver, based on backtrack search, that exploits structural properties of a problem instance, such as identifying backdoor variables [4] and exploiting them to decompose the problem into tractable components. *Third*, we introduce four reformulation techniques to reduce the cost of problem solving. These techniques are (1) reformulating the BID problem from a counting problem to a satisfiability one, (2) reducing the domains size of variables in the scope of a global constraint that we identify and characterize, (3) relaxing the satisfiability problem into a matching problem, (4) using symmetry to generate efficiently all possible solutions of the relaxed version of the original BID counting problem. *Fourth*, as we introduce each reformulation technique, we also discuss its application to general CSPs. *Fifth*, we evaluate the benefits of 3 of our reformulations on the BID problem, showing that we can now solve instances involving 206 buildings while the problem solved by Michalowski and Knoblock included only 34 buildings.

This paper is structured as follows. Section 2 positions our adopted perspective on reformulation. Section 3 describes the new CSP model and custom solver for the BID problem. Sections 4, 5, 6, and 7 describe our reformulations of the BID problem and their utility for general CSPs. Section 8 evaluates our techniques on real-world BID instances. Finally, Section 9 describes related work and concludes the paper.

2 Background

Choueiry et al. [5] characterized 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 reformulation is to 'simplify' problem solving, where the benefit of the 'simplification' and other effects of the reformulation are clearly articulated in the particular problem-solving context. The reformulation techniques discussed in this paper 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 consistent solution or 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.

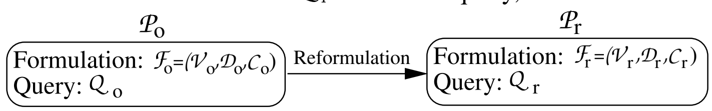

Fig. 1. The general pattern of a CSP transformation.

3 Modeling and solving the BID problem as a CSP

The task is to assign possible addresses to the buildings that appear in a satellite image. Each address consists of the combination of a street name and a number. The names of the streets are provided by a map and the positions of the buildings are extracted from a satellite image. Thus, we know the street names and the positions of the buildings, but we do not know the addresses of the buildings or, for buildings located on corners, on which street the buildings are located. The addresses can be partially retrieved from a variety of data sources such as a phone books, gazetteers, or property records. We generically refer to the addresses given as input as phone-book addresses regardless of their actual source. Figure 2 shows a BID instance with 10 buildings. The set of phone-book addresses may be *incomplete*, that is, there could be fewer addresses than there are buildings in an image. However, we assume that the reverse does not hold, that is, every phone-book address must be assigned to a building on the image. A solver must infer addresses for buildings that do not have an address in the phone book. In addition to the phone-book addresses, we may have information about street-numbering schemas used in a given region in the world, such as the 100-block increment in the addresses across street intersections used in the US or the red-black numbering used in Italy. Also, we may know the exact address of one or more *landmarks*, such as the residence of the Prime Minister in London.

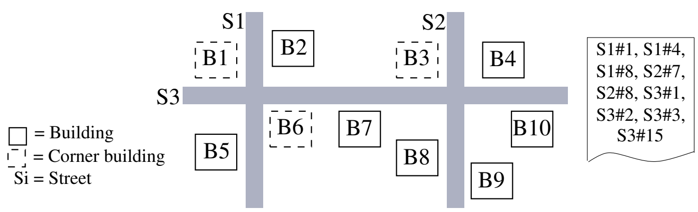

Fig. 2. An example of the building-identification problem.

3.1 A new constraint model

Below we describe the variables and constraints in our CSP model of the BID problem. Our model uses three types of variables: *orientation*, *corner*, and *building*. In general, there are four *orientation variables*. These Boolean variables determine the global orientation properties of the map. 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. *The corner buildings are natural 'backdoors' [4] in the constraint network*: once the solver assigns values to all corner buildings, the network degenerates into a set of chains (corresponding to buildings along street segments) that can be solved in a backtrack-free manner. Thus our solver instantiates corner variables 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* are binary constraints that apply to the pair of variables of each corner building, namely, the corner variable (which determines the street), and the building variable (which determines the address on the street). It reinforces 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.

Our new model improves the original one proposed in [1] as follows. The number of variables for non-corner buildings is reduced by half, reducing number of variables between 37% and 43% in our test cases. Domains of the building variables in [1] were enumerated and upper bounds chosen arbitrary. They are represented as intervals with potentially infinite bounds in the new model. We reduced constraint arity from four to two for parity constraints, and from six to three for ordering constraints. Corner constraints are new and allow early decomposition of the problem. Grid constraints are also new and allow a more precise modeling of the real world. Interestingly, we show, in Section 6, that in the absence of grid constraints, the BID problem is tractable. The tractability of the BID problem in the presence of grid constraints remains an open question. Thus, modeling the BID problem as a CSP remains a pertinent approach because it gives us the flexibility to represent arbitrary constraints such as grid constraints and other street-addressing schemas used around the world.

3.2 A custom backtrack-search solver

Our custom solver, written in Java, is a backtrack-search procedure. We adapted the conflict-directed backjumping mechanism MAC-CBJ of [6] to handle constraints of any arity with nFC3, a look-ahead strategy for non-binary CSPs [7], yielding nFC3-CBJ. Key to the solver's success are the domain representation and the variable ordering. Domains of building variables are represented as a list of intervals, where an interval is a sequence of values. This representation allows us to restrict propagation to the boundaries of the intervals, as in bound consistency, whenever possible, and iterate over the individual values only when necessary. Using intervals with arbitrary large bounds is crucial when the phone book is incomplete and the smallest or largest address number on a given street is not known. Variables are ordered as follows: building and corner variables corresponding to landmark buildings, orientation variables, corner variables, then building variables. Because corner variables are backdoor variables, *satisfiability can be determined without instantiating the building variables*, which are instantiated only when full solutions are sought. Further, instantiating the backdoor variables (corner variables) decomposes the problem into chains, one for each entire street.

4 Query reformulation

Michalowski and Knoblock [1] searched for all solutions in order to retrieve for each building on the map the set of acceptable addresses. *When the phone book is complete*, the problem has few solutions. Our solver, but not the one in [1], can easily find all solutions for *all* real-world examples we tested. *When the phone book is not complete*, the number of solutions quickly increases. The sheer number of solutions to be enumerated forced us to reconsider the task and reformulate the original query as explained below.

4.1 Per-variable solutions

Finding all solutions of a CSP is $O(d^n)$ where n is the number of variables and d is the maximum domain size. In practice, this process is prohibitively expensive. We consider the situation where we do not need to find all solutions, but only the values that each variable takes in any solution. We call this problem *finding the per-variable solutions*[3]. Thus, we reformulate the query from $\mathcal{Q}_{\mathrm{o}}$= enumerating all solutions, to $\mathcal{Q}_{\mathrm{r}}$= finding the per-variable solutions, where $\mathcal{Q}_{\mathrm{r}}$ is "$\forall\, V_i, x \in D_{V_i}$, find if $\mathcal{P}_{\mathrm{o}} \wedge (V_i \leftarrow x)$ is satisfiable" as illustrated in Figure 3. This query changes the complexity class of the problem from a counting problem to a satisfiability one.

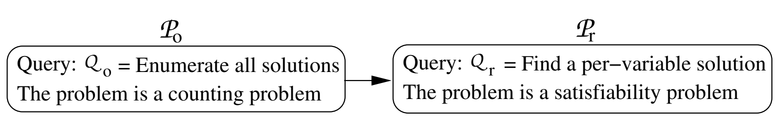

Fig. 3. Reformulation for the per-variable solution query.

Algorithm 1 tests for every variable-value pair (V_i, x) if the CSP with $V_i \leftarrow x$ is solvable. When it is, x is added to the data structure returned by the algorithm. Algorithm 1 returns the set of variables along with all their values that appear in a solution.

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] Formally, this query corresponds to finding the minimal CSP. It is also equivalent to the inverse consistency property introduced in [8], and to relational (1,$|\mathcal{C}|$)-consistency defined in [9].

Input: $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$
Output: S, a per-variable solution

1 **foreach** $V_i \in \mathcal{V}$ **do**
2     $S[V_i] \leftarrow \emptyset$
3 **end**
4 **foreach** $V_i \in \mathcal{V}$ **do**
5   **foreach** $x \in D_{V_i}$ **do**
6   **if** *$\mathcal{P}$ with $V_i \leftarrow x$ has a solution* **then**
7             $S[V_i] \leftarrow S[V_i] \cup \{x\}$
8   **end**
9   **end**
10   **if** $|S[v]| = 0$ **then**
11   **return** *$\mathcal{P}$ has no solutions*
12   **end**
13 **end**
14 **return** S

Algorithm 1: Finding the per-variable solutions.

When the test in Line 6 is executed by finding a solution to the CSP, the values for the variables in the solution found can be collected, and excluded from future calls in the loops on Lines 1 and 5 thus reducing the number of loops[4]. In the BID problem, we are not able to exploit this improvement for the following reason. A variable-value pair in Algorithm 1 for the BID problem is a combination of a building and a street name and number. However, the satisfiability of the BID instance is determined, and search is terminated, after the assignment of the backdoor variables and without instantiating the building variables (see Section 3.2). The benefit of continuing search and generating solutions after the instantiation of the backdoor variables in order to exploit the above improvement remains to be assessed.

4.2 Application to relational (i, m)-consistency

In non-binary CSPs, in order to enforce higher level consistency than (generalized) arc-consistency, Dechter and van Beek [9]introduced *relational* (i, m)*-consistency* as the consistency of m non-binary constraints over every subset of i variables in the CSP. Dechter [10] 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, join the m constraints and project the result on each subset of i variables. The algorithm is not practical for large values of m, because the memory requirements for computing and storing a join of m constraints rises exponentially with the number of variables in the scopes of these constraints.

Algorithm 1 computes a minimal network, and 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 can easily generalize Algorithm 1 to consider sets of i variables (and all tuples in the Cartesian product of their domain) rather than a single variable (and a single variable-value pair). This extension would allow Algorithm 1 to produce the same results as $RC_{i,m}$. The memory requirement rises exponentially with i, which quickly becomes impractical, but remains more efficient than $RC_{(i,m)}$ whose space complexity is exponential in the size of the union of the m constraints scopes.

[4] This improvement was suggested by an anonymous reviewer.

5 Domain reformulation using symbolic values

If the phone book is incomplete, we must infer the missing numbers to add to the variables' domains. Michalowski and Knoblock [1] proposed to enumerate all numbers between 1 and the largest address that appears on the street. 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. We propose a reformulation of the variables domains that reduces their size using symbolic variables, thus solving both problems.

5.1 Symbolic values in the BID problem

Assume we have, on the even side of a street S, the set of buildings B_S=$\{B_1$, B_2, . . ., $B_5\}$, the set of phone-book addresses of even parity P_S=$\{$S#12, S#18$\}$, and the range of address numbers [2,624]. Any assignment cannot use more than 3 numbers in each of [2,12), (12,18), and (18,624]. Using symbolic values to represent an address in a solution, we replace the domain [1,624] of each variable B_S 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. This process allows us to choose arbitrarily large bounds on a given street. Figure 4 illustrates this transformation. More

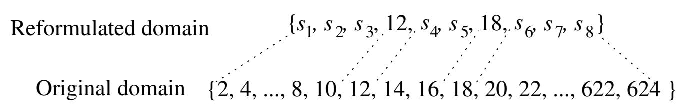

Fig. 4. Domain reformulation for the building-identification problem.

generally, when [min,max] is the range of address numbers on the considered 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 minimum($|B_S|$-$|P_S|$, $\lfloor\frac{(i_2-i_1)-1}{2}\rfloor$) addresses. Below we introduce ALLDIFF-ATMOST as a global constraint useful in such situations and 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.

5.2 The ALLDIFF-ATMOST global constraint

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 $d \subseteq D_{V_i}$ for $i \in [1, n]$, $k \in \mathbb{N}$, and $k \leq |d|$, requires that (1) all variables take different values 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[5].

[5] Many definitions of the ATMOST constraint exist (e.g., ECLiPse and on page 148 of [11]). Our definition of ALLDIFF-ATMOST allows us to express a situation of interest to resource allocation problems where our reformulation can be used to reduce the domain size.

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*←5,* V_2*←2,* V_3*←7 and* V_4*←4 satisfies the constraint.*

We can express the above described situation for the BID problem as ALLDIFF-ATMOST(B_S, k_a, (i_1, i_2)) with k_a=minimum($|B_S|$-$|P_S|$, $\lfloor\frac{(i_2-i_1)-1}{2}\rfloor$).

5.3 ALLDIFF-ATMOST reformulation

Our reformulation of the domains of the variables in a ALLDIFF-ATMOST constraint is theorem constant, in the sense that solutions to the reformulated problem map to solutions to the original problem [12]. 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 $\{s_1$, $s_2\}$ ∪ $\{$sites in non-volcanic areas$\}$ where s_1, s_2 are different and range over sites with volcanic activities.

This reformulation operates on the problem formulation and affects, strictly speaking, both the ALLDIFF-ATMOST constraint and the domains of the variables in its scope, see Figure 5. 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 $\mathcal{A}$ have been reformulated according to Equation (1). Replacing d with k symbolic values reduces the domains sizes by $|d| - k$, which is useful when d is large or infinite.

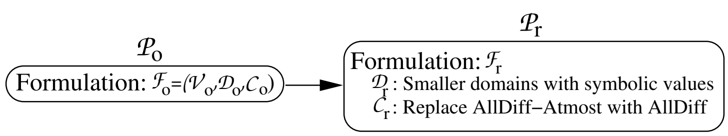

Fig. 5. The reformulation of ALLDIFF-ATMOST.

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 problem, 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, as is the case in a BID instance with a complete phone-book. While this double representation works for constraint propagation, using it during backtrack search requires further investigation.

5.4 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 total ordering constraints between the symbolic values: $s_1 < s_2 < \ldots < s_k$.
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. 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. Figure 6 illustrates this transformation.

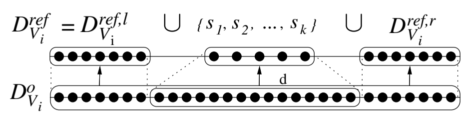

Fig. 6. ALLDIFF-ATMOST reformulation for totally ordered domains.

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

In the BID problem, we use this particular form of the reformulation of the ALLDIFF-ATMOST on the building variables, which have totally ordered domains.

6 Problem relaxation by constraint removal

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* (page 107 in [11]) and theory approximation [13]; in mathematical programming, linear relaxation of integer programs, Lagrangian relaxation [14], and the cutting-plane method. Below, we show that removing the grid constraint from the BID problem yields a tractable problem that is a tractable necessary approximation of the BID problem.

6.1 A tractable necessary approximation of the BID problem

We describe a construction to efficiently solve the BID problem in the absence of grid constraints by finding a maximum matching in a bipartite graph. We first recall some terminology. 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 for all $v \in V$ 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 for all $v \in V$ 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. A matching that saturates S can be computed in polynomial time [15].

Given an instance of the BID 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 and set its match count to 1. For each street σ in the problem, add two vertices s_{odd} and s_{even} to S, one for each side of the street. Set the match count of each s_i to the number of phone-book addresses on street s with parity i. 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.) Figure 7 shows the construction of G for the map in Figure 2 where we assume that odd numbers appear on the North and West sides of the street. We can show that a matching in this graph that saturates S corresponds to a satisfactory assignment of streets to corner buildings[6]. We find a maximum matching using an $O(n^{5/2})$ algorithm by Hopcroft and Karp [16] after replacing each vertex in the bipartite graph by as many vertices as its match count.

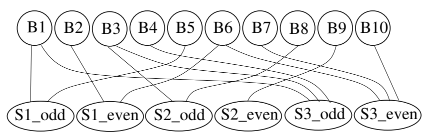

Fig. 7. Graph construction for Figure 2.

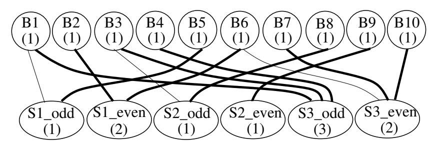

Fig. 8. A saturating matching for Figure 7.

Figure 8 shows a saturating matching for the graph of Figure 7, where the edges of the matching are darkened and the numbers in parentheses indicate the match count. This matching determines the satisfiability of the relaxed BID problem, and yields assignments to all corner variables in the corresponding CSP. For a complete solution, we still need to instantiate the building variables, which can be done in linear time because the constraint network becomes a set of chains after the instantiation of the backdoor (corner) variables. While the matching approach is powerful, it does not model the grid constraint. The tractability of the problem with grid constraints remains an open question.

6.2 Relaxing resource allocation problems

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 resource allocation problem may be complex (and likely intractable). However, we may sometimes be able to identify those constraints that, when removed, reduce the original problem into the problem of finding a matching in a bipartite graph that saturates one of the two partitions as described above. Figure 9 illustrates this relaxation.

[6] The matching must saturate S because the BID problem assumes that all addresses in the phone book, whether complete or incomplete, must be assigned to a building.

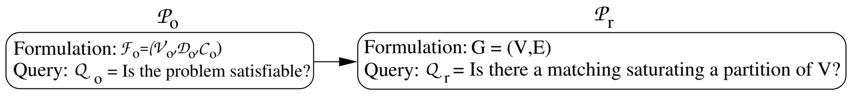

Fig. 9. Relaxing a CSP as a matching problem.

6.3 Using the relaxation in problem solving

We can use the above relaxation in four ways for the BID problem and for other applications that can be relaxed as a matching problem:

1. To solve problem instances that do not have the grid constraints, e.g. [1].
2. As a first preprocessing step to quickly rule out unsatisfiable instances, i.e. before Line 1 in Algorithm 1. Our experiments on the BID problem (not included here for lack of space) showed that this early preprocessing is effective only on tight problems.
3. As a second preprocessing between Line 5 and Line 6 in Algorithm 1, see Section 8.
4. As a lookahead mechanism when using search at Line 6 in Algorithm 1. We use the construction of [17] to filter out, from the domains of the future variables, those values that cannot yield a solution. As such, the relaxed problem appears as a (special version of the) all-diff constraints of [17], added to the problem as a *new* but redundant constraint to enhance propagation, see Section 8.

7 Generating solutions by symmetry

The set of solutions to the relaxed problem of Section 6 can be obtained by enumerating all maximum matchings using an algorithm such as the one proposed by Uno [18]. 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 [19]: *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[7] that there exists an $\mathcal{S}_j$ such that $M_j = M \Delta \mathcal{S}_j$. We generate $\mathcal{S}$ by first orienting G using the construction described by Hopcroft and Karp [16]. From the oriented graph, we enumerate the alternating paths by finding all EvAltP's, as defined by Berge [19]. We enumerate the AltCyc's from the strongly connected components in the oriented graph as described by Régin [17]. Thus, to store the information necessary to enumerate all alternating paths and cycles, and

[7] The proof is omitted for lack of space.

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[8].

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 10 (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 10 (b) show $M\Delta P$, Figure 10 (c) shows $M\Delta C$, and Figure 10 (d) shows $M\Delta(C \cup P)$.

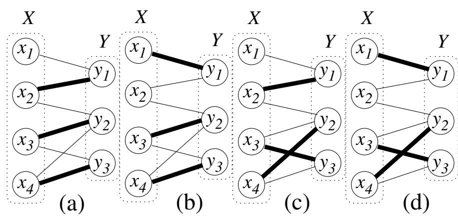

Fig. 10. Multiple matchings saturating Y.

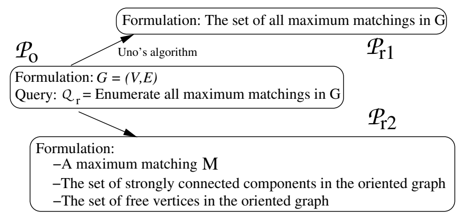

Fig. 11. Finding all maximum matchings.

Figure 11 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 has valuable properties which we do not fully exploit:

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 problem (similarly, to a resource allocation problem that has a maximum matching relaxation), we can generate every symmetric matching 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.

8 Experiments

We integrate our techniques in the flowchart shown in Figure 12, which implements the instruction in Line 6 of Algorithm 1. Table 1 describes the properties of the regions of the the city of El Segundo (CA), on which we ran our experiments. The number of calls refers to the total number of calls 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.

[8] An improvement suggested by an anonymous reviewer.

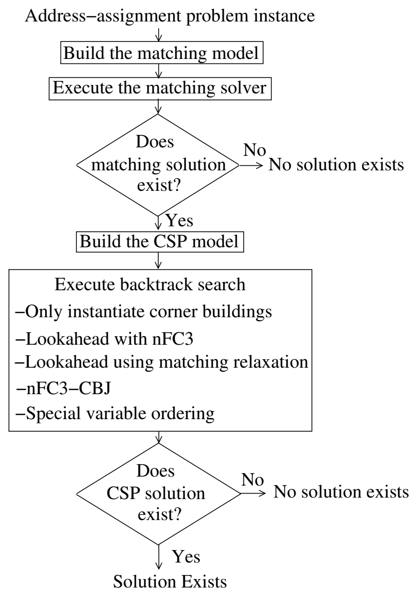

Fig. 12. Implementing Line 6 of Algorithm 1.

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. 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 domain reformulation. Table 2 shows the effect of domain reformulation by comparing the domain sizes and the cost of BT before and after reformulation. When the phone book is complete, the reformulation is not used as no ALLDIFF-ATMOST constraints exist. The advantage of the reformulation increases with the incompleteness of the phone book.

Effect of query reformulation. As stated in Section 4, 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 building-identification problem 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 7. 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%

9 Related work and conclusions

Reformulation has been applied to a wide range of CSP problems with much success. The literature also encompasses approaches to modeling, abstraction, approximation, and symmetry detection[9]. Nadel studied 8 different models of the n-Queens problem, some of which much easier to solve than others [20]. Glaisher proposed avoiding symmetry in the Eight Queens as far back as 1874 [21]. Holte and Choueiry provide a general discussion on abstraction and reformulation in AI including CSPs [22]. Razgon et al. [23] studied a class of problems that is similar to the one we investigate, and which they call Two Families of Sets constraints (TFOS). They introduced a technique for reformulating TFOS problems into network flow problems. Conceptually, the relaxed problem we study in Section 6 constitutes a special case of the TFOS problem.

An interesting feature of our work is the design of several techniques and their integration in a comprehensive framework for solving the BID problem while highlighting their usefulness for general CSPs. Also, our query reformulation facilitates a much wider use of relational consistency algorithms than was possible before. In the

[9] Some successful dedicated meetings are: Symposium on Abstraction, Reformulation and Approximation, Workshop on Modeling and Reformulation, Workshop on Symmetry in CSPs.

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.

References

1. Michalowski, M., Knoblock, C.: A Constraint Satisfaction Approach to Geospatial Reasoning. In: AAAI 2005. (2005) 423–429
2. Pickering, T.: Speech by Under Secretary of State T. Pickering on 06/17/1999 to the Chinese Government Regarding the Accidental Bombing of the PRC Embassy in Belgrade (1999)
3. van Beek, P., Chen, X.: CPlan: A Constraint Programming Approach to Planning. In: AAAI 1999. (1999) 585–590
4. Kilby, P., Slaney, J., Thiébaux, S., Walsh, T.: Backbones and Backdoors in Satisfiability. In: AAAI 2005. (2005) 1468–1373
5. 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
6. Prosser, P.: MAC-CBJ: Maintaining Arc Consistency with Conflict-Directed Backjumping. Technical Report 95/177, Univ. of Strathclyde (1995)
7. Bessière, C., Meseguer, P., Freuder, E., Larrosa, J.: On Forward Checking for Non-binary Constraint Satisfaction. In: CP 1999. (1999) 88–102
8. Freuder, E., Elfe, C.: Neighborhood Inverse Consistency Preprocessing. In: AAAI 1996. (1996) 202–208
9. Dechter, R., van Beek, P.: Local and Global Relational Consistency. Journal of Theoretical Computer Science (1996)
10. Dechter, R.: Constraint Processing. Morgan Kaufmann (2003)
11. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall (2003)
12. Giunchiglia, F., Walsh, T.: A Theory of Abstraction. Artificial Intelligence **57**(2-3) (1992) 323–389
13. Selman, B., Kautz, H.: Knowledge Compilation and Theory Approximation. Journal of the ACM **43**(2) (1996) 193–224
14. Milano, M., ed.: Constraint and Integer Programming: Toward a Unified Methodology. Kluwer Academic Publishers (2004)
15. Gallai, T.: Über extreme Punkt- und Kantenmengen. Ann. Univ. Sci. Budapest, Eotvos Sect. Math. **2** (1959) 133–139
16. Hopcroft, J., Karp, R.: An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. SIAM **2** (1973) 225–231
17. Régin, J.: A Filtering Algorithm for Constraints of Difference in CSPs. In: AAAI 1994. (1994) 362–367
18. 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
19. Berge, C.: Graphs and Hypergraphs. American Elsevier (1973)
20. Nadel, B.: Representation Selection for Constraint Satisfaction: A Case Study Using n-Queens. IEEE Expert **5**(3) (1990) 16–24
21. Glaisher, J.: On the Problem of the Eight Queens. Philosophical Magazine **4**(48) (1874) 457–467
22. Holte, R.C., Choueiry, B.Y.: Abstraction and Reformulation in Artificial Intelligence. Philosophical Trans. of the Royal Society Sect. Biological Sciences **358**(1435) (2003) 1197–1204
23. Razgon, I., O'Sullivan, B., Provan, G.: Generalizing Global Constraints Based on Network Flows. In: Workshop on Constraint Modelling and Reformulation. (2006) 74–87