2002

Constraint Modeling and Reformulation in the Context of Academic Task Assignment

Robert Glaubius
University of Nebraska-Lincoln, glaubius@cse.unl.edu

Berthe Y. Choueiry
University of Nebraska - Lincoln, choueiry@cse.unl.edu

Constraint Modeling and Reformulation in the Context of Academic Task Assignment

Robert Glaubius and Berthe Y. Choueiry

Department of Computer Science and Engineering, 115 Ferguson Hall,
University of Nebraska-Lincoln, Lincoln NE 68588-0115
{glaubius | choueiry}@cse.unl.edu

Abstract. We discuss the modeling and reformulation of a resource allocation problem, the assignment of Graduate Teaching Assistants to courses in the University of Nebraska-Lincoln Computer Science Department. We formulate this problem as a non-binary Constraint Satisfaction Problem (CSP) and provide a new convention for consistency checking to deal with the over-constraindness of the problem and the practical requirements of our application. We introduce a new decomposable non-binary constraint, which we call confinement constraint, and describe its relevance in practical settings. We discuss the reformulation of confinement constraints and equality constraints into an equivalent network of binary constraints. Empirical evaluations on three sets of real-world data demonstrate the benefit of such reformulations in reducing the processing time to reach equivalent solutions.

1 Introduction

Constraint Satisfaction has emerged as a powerful paradigm for modeling and solving large combinatorial problems. Scheduling and resource allocation are some of the earliest application areas of this technology [4, 3]. In this document, we discuss a specific application of constraint satisfaction techniques to a real-world application. This is the assignment of Graduate Teaching Assistants (GTA) to courses in the Department of Computer Science and Engineering of the University of Nebraska-Lincoln. The idea for this particular application was found on the web page of Rina Dechter at the University of California, Irvine. This application is in fact a critical and arduous responsibility that the department's administration has to handle every semester.

Typically, each semester a pool of 25 to 40 GTAs must be assigned as graders or instructors to the majority of courses offered during that semester. In the past, this task has been performed by hand by several members of the staff and faculty. Tentative schedules were iteratively refined and updated based on feedback from other faculty members and the students themselves, in a tedious and error-prone process dragging over 3 weeks. It was quite common that the final hand-made assignments still contained a number of conflicts and inconsistencies, which negatively affects the quality of our program. For instance, when a course is assigned a GTA that has little knowledge of the subject matter, the course's

instructor has to take over a large portion of the GTA's job and the GTA has to invest considerable effort to adjust to the situation. Moreover, students in the course may receive diminished feedback and unfair grading.

Our efforts in modeling and solving this problem using constraint processing techniques have resulted in a prototype system under field-test in our department since August 2001 [7]. This system has effectively reduced the number of obvious conflicts, thus yielding a commensurate increase in course quality. It has also decreased the amount of time and effort spent on making the assignment and gained the approval and satisfaction of our staff, faculty and student body.

In this paper, we describe the modeling of this application, which involves a number of non-binary constraints. In order to deal with the fact that the problem is always over-constrained, we propose a new convention for consistency checking that allows variables to be assigned a `null` value. Further, we identify a new type of non-binary constraint that we call *confinement constraint* and demonstrate to be network decomposable [8]. We introduce the confinement and equality constraints, give their reformulation as a network of binary constraints, and evaluate the effects of these reformulations on the performance of backtrack search using real-world data sets we have collected. We argue that, as a result of our particular consistency-checking mechanism, the reformulation of the confinement, equality, and all-different constraints into networks of binary constraints yields exactly the same tree and the same domain reductions for backtrack search while substantially reducing the CPU time.

This paper is structured as follows. Section 2 defines a constraint satisfaction problem. Section 3 describes the GTA assignment problem. Section 4 discusses its formulation as a finite, non-binary CSP. Section 5 discusses the reformulation of confinement and equality constraints into equivalent networks of binary constraints. Finally, Section 6 describes our experiments and Section 7 concludes this paper.

2 Definitions

A Constraint Satisfaction Problem (CSP) is a triple, $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$, where $\mathcal{V} = \{V_1, V_2, \ldots, V_n\}$ is a set of variables, and $\mathcal{D} = \{D_{V_1}, D_{V_2}, \ldots, D_{V_n}\}$ is the set variable domains, such that D_{V_i} is the domain of variable V_i. When the domains are finite, the CSP is said to be finite. $\mathcal{C} = \{C_i, C_{j,k}, \ldots, C_{i,j,\ldots,m}, \ldots C_n\}$ is a set of constraints such that $C_{i,j}$ indicates a constraint between variables V_i and V_j. Such a constraint (i.e., C_{V_i,V_j}) specifies the set of allowable tuples of values that can be assigned to the variables V_i and V_j. A constraint can be specified in extension (allowable tuples are enumerated) or in intension (a predicate determines whether a given tuple is acceptable). For a given constraint $C_{i,j,\ldots,m}$, the set of variables $V_i, V_j, \ldots, V_m$ is the constraint's *scope* and the cardinality of this set is the constraint's *arity*.

Formally, solving a CSP $\mathcal{P}$=($\mathcal{V}$, $\mathcal{D}$, $\mathcal{C}$), requires finding an assignment to each $V_i \in \mathcal{V}$ from its domain D_{V_i} such that all constraints are simultaneously satisfied. Much effort in research on finite CSP is focused on binary constraints, in which

only constraints with arity ≤ 2 are considered (notable exceptions are [10, 1, 6]). We use both binary and non-binary constraints to model our application.

3 Problem definition

The GTA assignment problem can be stated as follows. In a given semester, given a set G of graduate teaching assistants, a set $\mathcal{V}$ of courses, and a set of constraints that specify allowable assignments, find a *consistent* and *satisfactory* assignment of GTAs to courses. Since in practice this problem is systematically over-constrained, some courses in the final assignment may not have GTA assigned to them. An assignment is said to be 'consistent' when no constraint is broken even if some courses have not been assigned a GTA.

Since the goal of the department is to ensure GTA support to every course that needs one, we measure the degree of 'satisfaction' of a solution primarily by the number of courses that are assigned a GTA, and, secondarily by its quality in terms of some preference criteria that we discuss in Section 3.4.

3.1 Course features

The courses in this problem can be described by a set of features. These features determine some of the conditions under which a given GTA can be assigned to a particular course. They are: course type, course duration, meeting time and course weight. Below we describe these features.

We identify 3 distinct types of courses offered: lectures, labs, and recitations. These courses may require a GTA as either an instructor or grading assistant, or both. Labs and recitations require GTAs as instructors, while lectures usually require GTAs as grading assistants. Few exceptions exist and are explicitly stated by the administrator in charge of the assignment.

The duration of a course within a semester may vary. There are three possible intervals a course may span: the full semester, the first half of the semester, or the second half. The time spans of the half semesters are disjoint.

We consider courses that have a fixed, predetermined meeting time, specified by time of day and days of the week. Meeting times are significant in cases where a GTA is required as an instructor: a GTA cannot be assigned to teach a given course A if he or she is enrolled in another course that overlaps in time with A.

Prior to assignment, each course is assigned an expected weight. This can a real value in the interval [0,1] but is usually a discrete value (i.e., 0, 0.25, 0.5, or 1). A course A with $weight(A) = 0$ indicates that no GTA should be assigned to A. Non-zero weight values are discussed in more detail in Section 3.2 where we introduce the features of GTAs.

3.2 GTA features

In the previous section we gave an overview of the attributes of courses. This section discusses the relevant features of GTAs. These features are the GTA's

enrollment status, ITA certification, type of assistantship, and individual course preferences.

A GTA's enrollment status is the set of courses that he or she is enrolled in during the semester. This feature is taken into consideration for two reasons. The first is to avoid time conflicts with courses that the GTA is scheduled to teach. The second purpose is to avoid situations in which a GTA is scheduled to teach or grade for a course that he or she is enrolled in.

International Teaching Assistant (ITA) certification is required by law for an international GTA to instruct a course. GTAs without ITA certification may only be assigned as grading assistants, while ITA certified GTAs may be assigned as either grading assistants or instructors.

GTAs may be awarded either half or full teaching assistantships. We associate a capacity max_g with each GTA. A GTA with a half TA-ship has capacity $max_g = 0.5$ and a GTA with a full TA-ship has capacity $max_g = 1$. A GTA's capacity indicates the maximum course weight he or she can be assigned at any point in time during a semester.

Since the beginning of our effort, we introduced a new attribute into the data collection process. This attribute is the GTA's preference for each course offered. These preferences are integer values in $\{0, 1, \ldots, 5\}$. When a GTA assigns preference 0 to a course, this indicates that the GTA should not be assigned to the course, while 5 expresses a strong preference of the GTA for the assignment.

3.3 Assignment constraints

In the previous two sections, we looked at features of individual courses and GTAs. In this section we will discuss some of the rules and guidelines that dictate whether a GTA can or cannot be assigned to certain courses. Some of these constraints were alluded to in the discussion of features. For instance, a GTA must be ITA certified before he or she can teach a course.

Other rules we have already hinted at avoid conflict between enrollment and assignment, such as avoiding assignments that conflict in time with courses that a GTA is enrolled in. We also want to prevent any GTA from being assigned to courses that he or she is taking, or that he or she has zero preference for. Finally, we want to prevent any GTA from being assigned a heavier workload than he or she has capacity for.

In addition to the previous constraints, several guidelines are also followed in practice, which we integrate into our model as constraints. For instance, the department sometimes wants to assign to some arbitrary set of courses the same GTA. Another guideline is concerned with labs and recitations. Some lecture courses have several sections, for example, there may be four different sections of Introduction to Computer Science. Each of these lecture sections may have several labs or recitations associated with it. Often, the weight of these labs is too much for one GTA, but we would like to keep the number of GTAs assigned to a set of these labs and recitations relatively small, e.g. two or three GTAs for five labs. Further, these GTAs should not be assigned to courses other than these

labs or recitations. Such a strategy allows these GTAs to focus their preparation on a group of similar courses.

3.4 Solution quality

Above, we formalized the GTA assignment problem in terms of finding a consistent and satisfactory assignment of GTAs to courses. The features of courses and GTAs, and the rules and guidelines for assignments give a sense of what it means for an assignment to be consistent. We have briefly addressed what constitutes a 'satisfactory' assignment, or its quality. In practice, it is often impossible to find an assignment to every course. In the context of our application, the natural criterion for a satisfactory assignment is to maximize the number of courses covered. For two assignments that cover the same number of courses, we further discriminate between them by choosing the one of highest quality, obtained by a combination of the value of the preferences in each assignment. We experimented with both maximization of the geometric mean of preferences and maximization of the minimum preference value in an assignment. Due to the similarity of the performance of the two measures in practice, we will discuss results only for the former.

4 The GTA assignment problem as a CSP

We formulate the GTA assignment problem as a constraint satisfaction problem. We express the courses as variables, the available GTAs as variable domains, and the rules and guidelines discussed in Section 3.3 as constraints.

In this section we first recall our convention for consistency checking in the context of over-constrained problems then the various constraints we have identified for our application. Our model includes four types of unary constraints, one type of binary constraint, and three types of non-binary constraints.

4.1 Consistency checking

Since in our application, problems are over-constrained, some variables in a solution may not be assigned a value. We choose to handle this situation in backtrack search by assigning the value `null` to such a variable and allow the search to proceed beyond the variable. Therefore, we modify the convention of constraint checking as follows. We say that a given tuple $\langle v_1, v_2, \ldots, v_k \rangle$ for k variables $V_1, V_2, \ldots, V_k$ is consistent with the constraint $C_{V_1,V_2,\ldots,V_k}$, when all non-null v_i in the tuple satisfy the constraint, regardless of whether some of the values in the tuple are actually `null`[1]. If we did not allow `null` values, then backtrack search would not be able to proceed to the remaining future variables. In practice, this convention results in the pruning power of nFC2 on the non-binary formulations

[1] Note that, in terms of relational algebra, this convention is equivalent to using the outerjoin of the relations defined by the constraints.

of confinement, equality, and `mutex` constraints to collapse to that of FC on the binary decomposition of these constraints.

Strictly speaking, our problem is not a CSP in that we do not require all variables in a solution to be instantiated[2]. In Max-CSP [5], one tries to find a complete assignment that minimizes the number of violated constraints. However, the practical requirements of our application dictate that we minimize the number of un-instantiated variables, not counting the number of constraints violated. Other strategies for dealing with over-constrained problems include the distinction between hard and soft constraints. Again, in our application, such a distinction is not made in practice. Although it *may* be possible to theoretically reduce our problem to some of the above listed general problems, we are not compelled to do so in order to remain as faithful as possible to the practical requirements of the application.

4.2 Constraint types

The unary constraints used in this application are ITA certification, enrollment, overlap, and zero preference constraints. ITA certification constraints and overlap constraints are placed on every course that requires a GTA as an instructor, while enrollment and zero preference constraints are placed on all courses. The enrollment constraint prevents a GTA from being assigned to a course that he or she is enrolled in, while the overlap constraint prevents a GTA from being assigned to teach a course that is held at the same time as a course he or she is enrolled in. Zero preference constraints prevent us from assigning a GTA to a course if he or she has zero preference for it.

The only binary constraint used in this application is a binary `mutex` constraint[3], which specifies that both courses in its scope cannot be assigned the same GTA, although both could possibly be assigned the `null` value[4]. In our model, `mutex` constraints are placed between any two courses that meet at overlapping times and require a GTA as an instructor. One is tempted to model this situation with a non-binary `mutex` constraint then use the powerful filtering algorithm proposed by Régin in [9]. This solution would be incorrect in our case since this filtering algorithm cannot handle `null` assignments to variables and returns failure on over-constrained problems, which prevents search from progressing.

We use three types of non-binary constraints: capacity constraints, equality constraints, and confinement constraints. We define and discuss each one in detail below.

Capacity constraints are n-ary constraints that prevent a GTA from being assigned a heavier workload than he or she should have. This maximum capacity is determined by the type of TA-ship of the GTA. Two capacity constraints are

[2] The formal definition of a CSP requires an assignment to *all* variables.

[3] a.k.a. `all-diff`, difference, or coloring constraint.

[4] Note that we also use binary `mutex` constraints in the reformulation of the non-binary confinement constraints.

used for each GTA. This is due to course durations throughout the semester. The courses from a given semester can be partitioned into three sets: courses that are held during just the first half of the semester, courses held during just the last half of the semester, and courses that are held throughout the semester. For a given GTA, one capacity constraint covers the courses that occur during the first half of the semester (including full-semester courses) and another capacity constraint covers the courses that occur during the second half of the semester (also including full-semester courses), as shown in Figure 1. The constraint for the specified GTA states that the sum of the weights of the courses in the scope of the constraint to which he or she is assigned cannot exceed the capacity of the GTA, and is expressed as follows:

$$\sum_{v \in Scope(C)} assigned(g, v) \cdot weight(v) \leq capacity(g)$$

where C is a capacity constraint specific to GTA g, and $assigned(g, v)$ is 1 if g is assigned to v, otherwise it is 0.

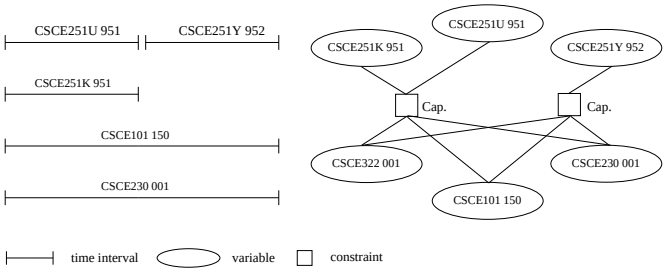

Fig. 1. Constraint network with two capacity constraints, one for courses in each half of the semester. Full semester courses are in the scope of both, while half semester courses are in the scope of just one capacity constraint.

If only one global constraint were used instead, it would be possible for a GTA to be assigned his or her maximum capacity in courses that occur during just one half of the semester. This is problematic, because during the other half of the semester, the GTA would not be teaching or grading. We chose to use two constraints, rather than increasing the complexity of checking a capacity constraint by checking the portion of a semester courses are held in.

Equality constraints take an arbitrary subset of courses as scope. Each of these courses should be assigned the same GTA. There may be a conflict between the equality constraint and a capacity constraint, so it is more accurate to specify the equality constraint as as preventing other GTAs from being assigned within its scope of the constraint once any variable in the scope is assigned a GTA. For instance, in Figure 2, we have five courses, three of which (CSCE150 002, CSCE150 003, and CSCE252 001) are the scope of an equality constraint. Figure 2 (b) demonstrates filtering during backtrack search with non-binary forward

checking (nFC) [2] when Ali is assigned to CSCE150 002. Note that Bob and Chang are filtered from the domains of the remaining variables in the scope of the equality constraint.

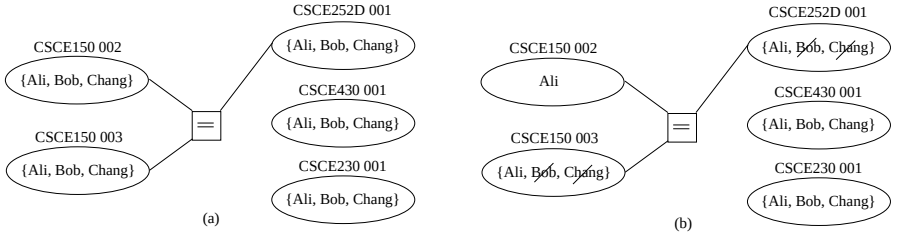

Fig. 2. (a) is an example of equality constraint use. (b) demonstrates filtering by FC during search after an assignment to CSCE150 002.

Confinement constraints allow us to specify that a GTA assigned to one or more courses in a given set S, called the *confinement set*, cannot be assigned to any course outside S, and vice versa. We use this constraint to prevent a GTA from being assigned outside the set of labs or recitations associated with a specific section of a course. Typically this is desirable when the section has several labs or courses and we want to allow the GTA to focus his or her efforts to the particular section. An example of such a constraint is illustrated in Figure 3 (a).

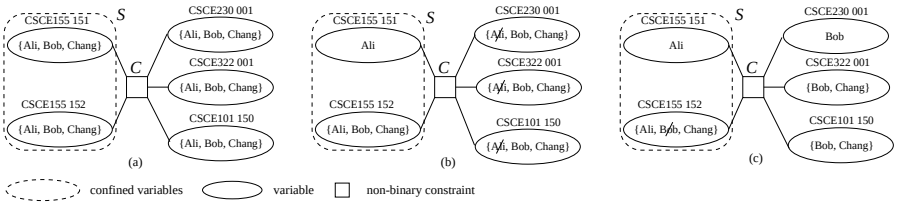

Fig. 3. (a) shows a constraint network with one confinement constraint. The constraint's confinement set is denoted by the dashed shape. (b) gives the filtering during search when Ali is assigned to CSCE155 151, and (c) gives the subsequent filtering when Bob is assigned to CSCE230 001.

The scope of a confinement constraint is the set of all courses that overlap in time. GTAs assigned to courses in S cannot be assigned outside of S. For instance, in Figure 3 (a), all courses are subject to a containment constraint C. C's confinement set S contains two variables, CSCE155 151 and CSCE155 152. Figure 3 (b) and (c) demonstrates the effect of C during backtrack search with non-binary forward checking. In (b), once Ali is assigned to CSCE155 151, we filter Ali from the domains of the variables outside S. In (c), once Bob is assigned

outside of S, he is removed from the domains of the variables in S. This new type of constraint has in fact a wide applicability and can be advantageously used to model other practical situations. For instance, in a system for scheduling workers to machines in a factory setting, we could define confinement sets to contain machines (variables) that are in the immediate vicinity or that require similar skills to operate.

Given that the scope of this confinement constraint is very large, it may prove expensive to check in practice. Therefore, it makes sense to attempt to reformulate it as a set of binary constraints. In section 5 we establish that confinement constraint is network decomposable and discuss its reformulation as an equivalent set of binary constraints.

5 Reformulation of constraints

Recently, researchers have been investigating whether it is advantageous to handle non-binary constraints directly or to first reformulate them as binary constraints [10, 1, 6]. There are three types of non-binary constraints in our application. In this paper, we address the reformulation of two of these constraints. These are the confinement constraint and the equality constraint.

In the discussion above, we expressed and formulated these constraints as non-binary constraints. We propose here to reformulate them through network decomposition into networks of binary constraints. In this section we describe the reformulation mechanism, the data, our experiments, and results.

5.1 Reformulation of equality constraints

We reformulate a k-ary equality constraint by replacing the constraint with a clique of $\frac{k(k-1)}{2}$ equality constraints between all variables in the scope of the constraint. This may seem to be an overkill as the same effect could be achieved with a chain of $(k-1)$ equality constraints as shown in Figure 4. However,

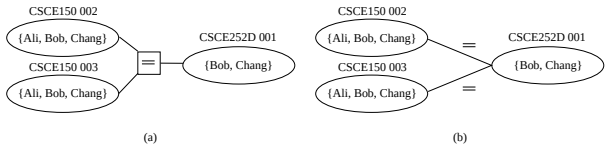

Fig. 4. (a) Shows a constraint network with one non-binary equality constraint (b) is one possible reformulation of this equality constraint. This scheme is not equivalent to the original constraint due to the possibility of assigning `null`.

since we admit `null` values, we cannot use the reformulation into a chain but have to reformulate the non-binary equality constraint by a clique of binary

constraints. The example of Figure 4 (b) illustrates this situation. Assume search first visits CSCE150 002, assigning Ali to it. The domain of CSCE252D 001 is wiped out. Instead of backtracking, search assigns this variable `null`. At this point, CSCE150 003 is not constrained to be assigned Ali, and may take a value inconsistent with CSCE150 002. In order to avoid such situations, we must place a binary equality constraint between every pair of variables in the scope of the original non-binary constraint.

5.2 Reformulation of Confinement constraints

The reformulation procedure for confinement constraints consists in replacing each confinement constraint C by a set of `mutex` constraints between every variable in S and every variable in $scope(C) \backslash S$. We can show that the initial constraint C is equivalent, in terms of the tuples entailed, to the set of projections of C on every pair of variables in $scope(C)$, which is the proposed reformulation. Thus, C is network decomposable. The reformulation transforms one confinement constraint into $|S| \cdot |scope(C) \backslash S|$ binary constraints, as shown in Figure 5.

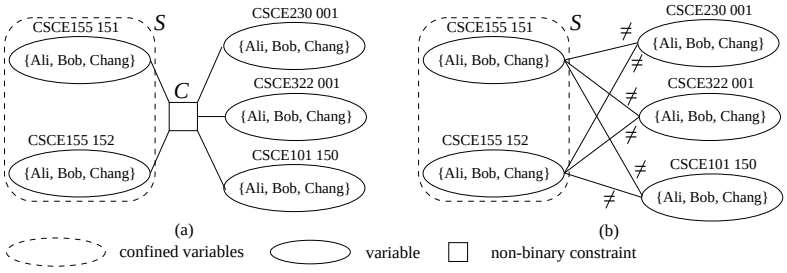

Fig. 5. (a) Shows the constraint network from figure 3. (b) is an equivalent CSP that represents the confinement constraint as a network of `mutex` constraints.

5.3 Expectations

Gent et al. compare [6] the performance of backtrack search with non-binary forward checking (nFC) [2] on non-binary constraints and to that of forward checking (FC) on the binary decomposition of the constraints and conclude that FC cannot visit less nodes than nFC1, nFC2, ..., nFC5. This can be justified intuitively as follows. Any of the nFCi with $i \geq 1$ has stronger pruning power on the non-binary constraint than FC has on the binary decomposition[5]. Therefore,

[5] nFC0 on the other hand has weaker pruning power on the non-binary constraint than FC has on the binary decomposition.

such an nFCi may annihilate the domains of future variables more quickly than nFC would, assuming the same variable ordering. Therefore, it may backtrack earlier and may end up visiting less nodes than FC.

In Section 4.1, we explained that our CSP model allows search to assign `null` to variables. This means that search does not backtrack when the domain of a variable is annihilated. Consequently, the above mentioned advantage of nFCi with $i \geq 1$ over FC on the binary reformulation does not necessarily hold. In fact, we can show that in the context of our convention for consistency checking, nFC1 and nFC2 on the non-binary equality, confinement and `mutex` constraints collapse to FC on their binary decompositions. In particular, the nodes explored and the domain reductions are exactly the same.

6 The data and experiments

We tested the effect of the reformulation of the confinement constraint on three data sets from the Spring 2001 and Fall 2001 and 2002 semesters in the Computer Science and Engineering department at the University of Nebraska-Lincoln. These data sets are summarized in Table 1[6].

	Data set		
	Spring 2001	Fall 2001	Fall 2002
Number of GTAs	25	34	31
Total number of courses	77	81	77
Lectures	44	47	45
Labs	24	24	24
Recitations	3	3	2
Half-semester	6	7	6
Number of equality constraints	3	3	10
Average arity	5	5.67	3.4
Number of capacity constraints	50	68	62
Average arity	63	58	65
Number of confinement constraints	12	16	14
Average arity	63	58	65
Average confinement set size	3.333	4.375	4.857

Table 1. Description of test data.

Our tests consisted of four experiments per data set. Each experiment is a combination of either a binary or non-binary model with static or dynamic

[6] The Spring 2001 data set did not include half and full TA-ship information; we assumed a default full TA-ship for all TA's. For the twenty GTAs who did not submit preferences for the Fall 2001 data set, we assumed a default preference of 3. This is also the case for 9 GTAs who did not submit preferences for Fall 2002

least domain variable ordering (SLD and DLD, respectively). We ran a depth-first backtrack search with a branch and bound strategy based on maximizing variables covered, breaking ties by choosing the solution that maximizes the geometric mean of GTA preferences. For each case, we used nFC2 proposed by Bessière et al. [2], taking advantage of the fact that all nFCi collapse to FC on binary constraints. Note that because of the way we handle consistency checking (see Section 4.1), search does not backtrack when the domain of a variable is annihilated.

6.1 Results of experiments

Each test was run for approximately one hour before halting, which was necessary because the problem is over-constrained. In Table 2, We report CPU time to find the best solution, the number of nodes visited (to best solution), the number of constraints checked (to best solution), and solution quality. We use the convention of Bacchus and van Beek [1] of incrementing the number of constraints checked by the arity of the constraint in question. We also report the length of the best solution (i.e., the number of variables assigned non-`null` values) and solution quality in terms of the geometric mean of GTA preferences. Note that we do not take into account the cost of the reformulation as it is polynomial in the number of variables of the CSP and the cost of solving (i.e., reformulation and search) is clearly dominated by the exponential cost of backtrack search.

6.2 Discussion

We compare the results in terms of the best solution found, the time spent on finding the solution, and the numbers of constraint checks and nodes visited during search. We examine the effect of problem modeling and variable ordering on solution quality.

For a given data set and a given ordering heuristic, both the non-binary and binary models resulted in the same solutions during the one-hour duration of the experiment. However, the best solution was found earlier in the binary model than the non-binary one and required fewer constraint checks. Indeed, the reduction of CPU time ranges from 8% (Spring 2001, DLD) to 22% (Fall 2002, SLD), with an average value of about 17%.

These experiments further show that the same number of nodes are visited to reach these solutions, corroborating that the search trees are equivalent and that nFC2 on the non-binary model collapses to FC on the binary model.

It is well known that dynamic variable ordering is superior to static ordering. Our results confirm this, as the best solution to every problem was found by dynamic variable ordering (DLD). In Fall 2001, the best solution found by DLD has a higher geometric mean of preferences than its SLD counterpart. In the other two data sets the final solutions found by DLD cover more variables than the SLD-ordered search.

CSP			Search running for one hour				Quality of best solution found			
Data	Vars	Vals	Order	Model	|Sol|	Capacity Left	CC	NV	Time (ms)	GeoMean
Spring 2001	69	25	SLD	binary	49	2.5	1208257106	514389	2463680	3.806217
				non-bin	49	2.5	1424663866	514389	2848450	3.806217
			DLD	binary	51	2.5	400736550	84423	614080	3.673231
				non-bin	51	2.5	400998214	84423	673020	3.673231
Fall 2001	65	34	SLD	binary	56	1	77809896	112	30630	3.167192
				non-bin	56	1	97854466	112	38970	3.167192
			DLD	binary	56	1	82827924	64	33360	3.354575
				non-bin	56	1	104189982	64	42630	3.354575
Fall 2002	71	31	SLD	binary	54	3.6	76231798	70	24570	3.564383
				non-bin	54	3.6	92933223	70	31520	3.564383
			DLD	binary	57	3.15	225355613	22560	255170	3.451227
				non-bin	57	3.15	252293613	22560	295790	3.451227

Table 2. Results of experiments. Vars and vals are the number of variables and values in each problem, respectively. Order is the ordering heuristic used: static least domain (SLD) or dynamic least domain (DLD). Model is either non-binary or the binary decomposition. |Sol| is the number of assigned variables in the solution. Capacity Left is the amount of cumulated GTA capacity left unassigned. CC is the number constraint checks. NV is the number of nodes visited. Time is CPU time in milliseconds, and GeoMean is the geometric mean of preferences in the solution.

7 Conclusions

In this paper, we report how a real-world problem was formulated as a Constraint Satisfaction Problem. We discuss the use of non-binary constraints for modeling and solving the problem. We introduce a new type of non-binary constraint useful in practical settings and provide its reformulation in a network of binary constraints. We report the results of experiments on two sets of real-world data and justify why theoretical predictions of the performance of search with forward checking are not applicable in our context.

We plan to enrich our data set not only with real-world data, but perhaps with some randomly generated ones. We also plan to refine this study by experimenting with various aggregation and decomposition techniques.

Acknowledgments

Marilyn Augustyn provided the GTA and course data and helped in the elicitation of the constraints. Christopher Hammack built the web-based user-interface for data collection currently used in the department. We are grateful for a reviewer's comments on a draft of this paper submitted to the ECAI 2002 Workshop on Modelling and Solving Problems with Constraints. This work has been partially supported by the Department of Computer Science and Engineering at UNL, the Constraint Systems Laboratory, and a Layman Award.

References

1. Fahiem Bacchus and Peter van Beek. On the Conversion between Non-Binary and Binary Constraint Satisfaction Problems. In *Proc. of AAAI-98*, pages 310–319, Madison, Wisconsin, 1998.
2. Christian Bessière, Pedro Meseguer, Eugene C. Freuder, and Javier Larrosa. On forward checking for non-binary constraint satisfaction. In *Principles and Practice of Constraint Programming (CP'99)*, pages 88–102, 1999.
3. Berthe Y. Choueiry and Boi Faltings. A Decomposition Heuristic for Resource Allocation. In *Proc. of the 11 th ECAI*, pages 585–589, Amsterdam, The Netherlands, 1994.
4. Mark Fox. *Constraint Directed Search: A Case Study of Job-Shop Scheduling.* Morgan and Kaufmann, Los Altos, CA, 1987.
5. Eugene C. Freuder and Richard J. Wallace. Partial Constraint Satisfaction. *Artificial Intelligence*, 58:21–70, 1992.
6. Ian Gent, Kostas Stergiou, and Toby Walsh. Decomposable Constraints. *Artificial Intelligence*, 123 (1-2):133–156, 2000.
7. Robert Glaubius. A Constraint Processing Approach to Assigning Graduate Teaching Assistants to Courses. Undergraduate Honors Thesis. Department of Computer Science and Engineering, University of Nebraska-Lincoln, 2001.
8. Ugo Montanari. Networks of Constraints: Fundamental Properties and Application to Picture Processing. *Information Sciences*, 7:95–132, 1974.
9. Jean-Charles Régin. A filtering algorithm for constraints of difference in constraint satisfaction problems. In *Proc. of AAAI-94*, pages 362–437, Seattle, WA, 1994.
10. Francesca Rossi, Charles Petrie, and Vasant Dhar. On the Equivalence of Constraint Satisfaction Problems. In *Proc. of the 9 th ECAI*, pages 550–556, Stockholm, Sweden, 1990.