A Constraint Logic Programming Solution to the Teacher Relocation Problem

Nagehan Ilhan, Zeki Bayram
Computer Engineering Department, Eastern Mediterranean University
Famagusta, Turkish Republic of Northern Cyprus
{nagehan.ilhan, zeki.bayram}@emu.edu.tr

Abstract — **In North Cyprus, teacher relocation in State schools is handled in a centralized manner. Teachers, if they wish to relocate, state which schools they wish to move to, and the authorities try to come up with a relocation scheme that maximizes the happiness of the teachers, without overlooking the needs of the schools involved. This is a classic discrete optimization problem, and requires exponential search for its solution. We handle this problem using constraint logic programming techniques, trying to minimize the search space wherever possible. We formulate the problem mathematically, model it in the ECL^{i}PS^e constraint logic programming language, and test our solution under various scenarios.**

I. INTRODUCTION

Many real-life problems need combinatorial search for their solution, and the solution time can grow exponentially with the size of the problem. The "teacher relocation problem," where each teacher states which set of schools would satisfy him/her, and an assignment of teachers to schools which tries to maximize the satisfaction of the teachers according to certain criteria is made, falls under this class of problems.

Constraint Logic Programming (CLP) has evolved to model and solve many hard real-life problems in recent years [1]. It is well suited for scheduling problems since it allows the formulation of the constraints of the problem in a high-level, declarative way. Variables of the problem are mapped to logic variables, domains of the variables are specified, and the constraints on the variables are stated declaratively [7]. An objective function on the variables involved is also specified. The goal is to find an assignment of values to the variables, without violating any constraints, in such a way that the value of the objective function is minimized.

In this paper, we define the "teacher relocation" problem, formulate it mathematically, model it in the ECL^{i}PS^e (ECRC Common Logic Programming System)[2] constraint programming language, and test our solution under various scenarios. Our results indicate that it is a viable solution which can be used in the real-life environment of North Cyprus to replace the manual method that is currently in use.

The rest of the paper is organized as follows. In section II we have the description and formal specification of the problem. Our CLP solution in ECL^{i}PS^e is presented in section III. Section IV contains the evaluation of our solution with simulated input data. In section V we have a brief coverage of related work, and section VI is the conclusion and future research directions.

II. DESCRIPTION, FORMAL SPECIFICATION AND TIME COMPLEXITY OF THE TEACHER RELOCATION PROBLEM

Each teacher who wishes to relocate can state two choices. An implicit, but undesirable third choice on the part of the teacher is to stay in his/her current school. Teachers who do not wish to relocate do not make any choices, and are not forcibly relocated.

Teachers state their choices in order of preference. A specific teacher's happiness is given very naturally by *order(teacher, assignment(teacher))*, where *order(t,s)* is the position (first, second etc.) of school *s* in the preference declaration of teacher *t*. Here, smaller numbers mean more happiness.

The goal of the system is to maximize the overall happiness of the teachers, which corresponds to minimizing the value of the objective function

$$\sum_{t \in TeachersWhoWishToRelocate} order(t, assignment(t)) \qquad (1)$$

subject to the constraint in each school that the number of teachers at the school never exceed its capacity. This is

modeled by the use of quotas, which specify the teacher deficiency (vacant positions) in each school before relocation is initiated. So for each school, the following constraint must be satisfied in a final solution:

$$incomingTeachers \leq outgoingTeachers + quota \quad \textbf{(2)}$$

Another obvious constraint is that a teacher cannot be assigned to more than one school, hence the function *assignment*.

For N teachers, the best case time complexity is N, where each teacher is assigned to his/her first choice without violating any constraints. For the worst case analysis, all possible combinations of assignments must be considered, which gives us 3^N.

III. FORMULATON OF THE PROBLEM AS AN ECL^{i}PS^e PROGRAM

Data Structures

To store the information of teachers, we use the structure *teacher(Teachername, ALSchool)* for each teacher. *Teachername* is the name of the teacher and *ALSchool* is the current school of the teacher. Each teacher name is assumed to be unique.

A teacher's choices are represented as *choice(Teachername, SchoolName, ChoiceNo, AppYear),* where *Teachername* is the name of the teacher, *SchoolName* specifies the name of the school which the teacher wants locate, *ChoiceNo* represents the precedence number of the teacher to locate to this school and *AppYear* is the application year of the teacher for relocation.

Information about schools is given by *school(Schoolname, Countyname, Quota, MaxCapacity),* where *SchoolName* is the name of the school, *Countyname* specifies the county which the school belongs to, *Quota* is the number of vacancies at the school, and *MaxCapacity* specifies the maximum number of teachers which the school can employ.

Note that not all the bits of information represented in the database are used in the solution that we present, but they are included to allow the possibility of different solutions in the future.

The general technique used

teacher-school pairs are assigned -1 or 0 to indicate whether *teacher* is assigned to *school*. -1 means that *teacher* is assigned to *school*, 0 means *teacher* is not assigned to *school*. This representation allows easy checking of the constraints that each teacher is assigned to exactly one school, and school quotas are not exceeded. Two techniques are used to prune the search space and generate results faster.

First, just any feasible assignment is generated, which gives us a baseline of "happiness" for teachers. If, during the optimization phase, this value is exceeded, that branch of the search space is pruned.

Second, the domains are increased gradually – if a solution is possible where all teachers are placed to their first choices, surely this will result in maximum teacher satisfaction. So a solution is attempted where only the first choices are considered. If no solution is possible, then the first two choices of teachers are considered. If still no solution is possible, then all three choices are considered (the last one being the implicit "stay in your current school" choice).

The top level predicate "solve"

The main predicate in the program is `solve`, given in Fig. 1. Its output parameter `Cost` is the minimum cost of the solution. The solution itself is printed on the screen. `solve` generates the teacher list using the database, and passes it on to the `increment_loop` predicate, which actually does the job. The second parameter of `increment_loop` denotes that only the first choices will be considered when the search begins.

```
solve(Cost):-
   findall(X,teacher(X,_),Teacherlist),
   increment_loop(Teacherlist,1,Cost),!.
```

Fig. 1: The "solve" predicate

The "increment_loop" predicate

The `increment_loop` predicate, given in Fig. 2, gets `Teacherlist` and `Choicenum` as input parameters and outputs the `Cost`. Predicate `willbelabeled` generates the list `Clist1`, which will be flattened by the `flatten` predicate to generate a list of the form [(*Teachername,Schoolname,R,Cost*),….] in its `Clist` output parameter. The domain of R is [-1,0] if *Schoolname* is either a school preferred by *Teachername* or it is his/her current school. Otherwise the domain of R is [0]. We chose -1, rather than 1, to represent a teacher being assigned to a school, because the built-in `indomain` predicate assigns values to variables in increasing numeric order, and in our labeling

predicate this would result in a teacher *not* being assigned to a school initially (0 would be tried before 1).

Then `each_teacher` predicate is called. It gets `Teacherlist` and `Clist` as input. It is the first constraint of the program which constrains a teacher to locate only to one school, since a teacher cannot exist in more than one school. In order to prevent the relocation of a teacher to more than one school , sum of all *R*'s of each teacher in `Clist` is constrained as to be -1.

$$SumofRs\ \#\ =\ -1 \qquad \textbf{(3)}$$

In the next step, the list of schools `Schoollist` is generated and passed as an input parameter to the `createoutoflist` predicate, together with `Teacherlist` and `Clist`. The predicate `createoutoflist` generates the `Outlist`, flattended into `Outoflist`, which has the same structure with `Clist`. However, the R value of a teacher becomes -1 in the school which he/she wants to vacate. `Outoflist` contains information about which schools teachers are moving out of.

```
increment_loop(Teacherlist,
               Choicenum,Cost):-
   willbelabeled(Teacherlist,Choicenum,
             [],Clist1),
   flatten(Clist1,Clist),
   each_teacher(Teacherlist,Clist),
   findall(Y,school(Y,_,_,_),Schoollist),
   createoutoflist(Teacherlist,Schoollist,
              [],Outlist,Clist),
   flatten(Outlist,Outoflist),
   findinginoutlist(Schoollist,Clist,
              [],Schoolinlist),
   findinginoutlist(Schoollist,Outoflist,
              [],Schooloutlist),
   quotaconstraint(Schoollist,Schoolinlist,
              Schooloutlist),
   solve1(RefCost),
   length(Teacherlist,LT),
   bb_min((our_labeling(Clist, 0, RefCost,
          Cost)), Cost,bb_options with
          [from:LT]),
   (nonvar(Cost),lastreplacement(Clist),!);
   (Choicenum1 is Choicenum+1,
              increment_loop(Teacherlist,
              Choicenum1,Cost)).
```

Fig. 2: The "increment_loop" predicate

Then `findinginoutlist` predicate is called two times. In first call, it takes `Schoollist` and `Clist` as input parameters and generates `Schoolinlist` as output. It keeps the information of the number of coming teachers to each school and has the structure [(*Schoolname*,*NumberOfComingTeachers*), …]. In second call of the `findinginoutlist` predicate, `Schoollist` and `Outoflist` are passed as input parameters to generate `Schooloutlist` which has the same structure with `Schoolinlist` but keeps the number of outgoing teachers for each school: [(*Schoolname*, *NumberOfOutgoingTeachers*)].

The `quotaconstraint` predicate is our second constraint in the program. It takes `Schoollist`, `Schoolinlist`, and `Schooloutlist` as parameters. It constrains each school such that the difference between the number of coming teachers and outgoing teachers to the school are smaller than or equal to the quota of the school.

$$(Incoming\text{-}Outgoing)\#<=Quota \qquad \textbf{(4)}$$

Then, the `solve1` predicate is called to get a baseline cost in its `RefCost` parameter. `solve1` just finds any feasible solution, without any consideration of optimality. We use this value to prune the search space when we make incremental assignments: if the current "cost" has already exceeded the reference cost, that branch of the search space is not explored any further. `Refcost` is passed as a parameter to `our_labeling` predicate.

Next we have a call to the minimization predicate *bb_min(Goal,Cost,Options). bb_min/3* is a built-in predicate which finds a solution of the *Goal* that minimizes the value of *Cost* [2]. The goal to be satisfied is `our_labeling(Clist,0,RefCost,Cost)`, where `Cost` is to be minimized. `Cost` is the cumulative happiness of teachers, as already discussed in section II, and is computed inside `our_labeling` predicate.

Then we have an OR (;) structure. If the program has already found the minimum cost, the `lastreplacement` predicate is called, which prints the teacher names, their assigned schools, as well as the rank of the school in the preference order of the teachers. If no solution was found, the `Cost` variable will be free, `Choicenum` is incremented by 1 and `increment_loop` is called recursively to try to find a solution by including one more preference of the teachers to the search space.

Labeling

Rather than relying on the built-in labeling predicate of ECL^{i}PS^e, we wrote our own `our_labeling` predicate given in Figure 3, which gave us the possibility of pruning the search space as labeling was being done.

```
our_labeling([],A,RefCost,A).
our_labeling([(H,S,R,X)|T],Acc,
              RefCost,Cost):-
   indomain(R),
   lk(H,S,R,X),
   Cost1 is Acc+X,
   unassignedTeachers(T,UL),
   Cost1+UL<=RefCost,
   our_labeling(T,Cost1,RefCost,Cost).
```

Fig. 3: The "our_labeling" predicate

The `our_labeling` predicate gets `Clist`, `Acc` and `RefCost` as input variables and outputs `Cost`. It firstly assigns a value to `R` by calling the `indomain` predicate. The `lk` predicate gets `H`(teacher name), `S`(school name) and `R` (0 or -1) as input parameters, and outputs `X`. If R = -1, then X = the order of preference of `S` by `H`. Otherwise `X`= 0.

`cost1` accumulates the cost ("cumulative happiness") of relocated teachers that have been assigned schools up to this point. We want to prune the search space at his point if the best achievable solution (`cost1` + number of teachers that have not been assigned to any school yet) is greater than or equal to `Refcost`. This is true, because in the best case, the remaining teachers will be assigned to their first choices, and their cumulative happiness will be equal to how many they are.

`unassignedTeachers(T, UL)` gives the number of unassigned teachers in `UL`. If `Cost1+UL>RefCost`, there is no need to continue to label the other teachers, and the labeling fails.

IV. RESULTS

It is possible to attempt to solve the teacher relocation problem without using our "incremental domain" approach. To see if there is any significant difference, we tried both approaches using simulated data, varying the number of teachers and the quotas of schools. The number of schools was fixed at 10 (admittedly an arbitrary figure, but representative of the number of elementary schools in North Cyprus).

Fig. 4 shows the number of teachers versus the CPU time in seconds to solve the problem for searching with incremental domains and without incremental domains, when empty quota's of schools randomly change from 1 to 5.

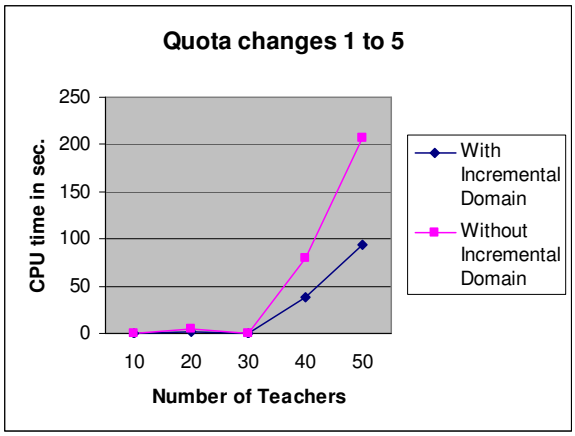

Fig. 4: Number of Teachers vs. CPU time while quota changes 1 to 5

It is clearly seen that with the incremental domain technique, the problem is solved much more quickly. This is especially true when the problem size becomes bigger in terms of the number of teachers.

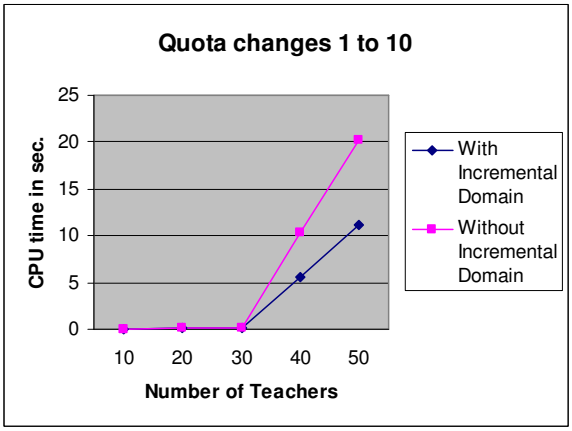

Fig. 5: Number of Teachers vs. CPU time while quota changes 1 to 10

As the quotas increase, the problem becomes easier, since teachers can then be assigned to schools without requiring that other teachers leave the school. This case is depicted in Fig. 5, where quotas range randomly from 1 to 10. In this case as well, the incremental domain approach shows a clear advantage.

The worst case of the problem occurs when we decrease quotas of the schools to zero (Fig. 6). In this case, the program is forced to tries almost all possibilities to find the solution and search space increases to its maximum level. The incremental domain approach still outperforms the other one, however

neither one can find a solution in a reasonable amount of time when the number of teachers is much more than 30.

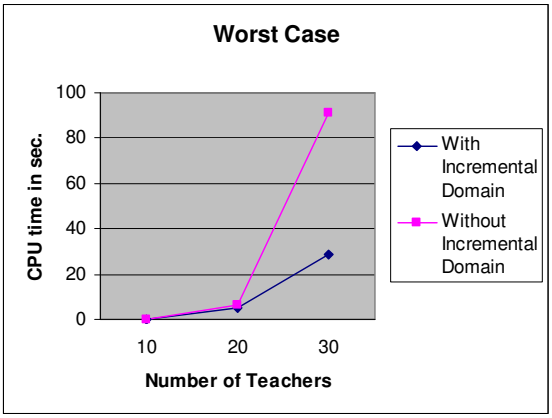

Fig. 6: Number of Teachers vs. CPU time while quotas are zero

V. RELATED WORK

Our literature search has not revealed any publication on the teacher relocation problem. Many other kinds of scheduling problems have been successfully solved with CLP, however. In [3] the authors present a methodology to solve a job-shop scheduling problem using constraint logic programming. They investigate a new strategy to find the optimal solution which involves step by step decreasing of the upper and lower bounds of the search space of the branch and bound evaluation function. In [4] authors describe a solution to the nurse scheduling problem by developing a technique to prune the search space which is based on reducing the variable domains. In [5], the authors investigate the efficiency of constraint programming for solving the nurse scheduling problem. In [6], the authors present a solution to university timetabling problem using the ECL^{i}PS^e constraint logic programming language by stating the constraints in the most suitable order. The authors in [8] use their own labeling to reduce the search space to find the optimal solution. In [9], a solution to sport tournament scheduling using the finite domain library of ECL^{i}PS^e is presented. A constraint-based depth-first branch and bound procedure is used to find an optimal solution in reasonable time, except in some situations. A solution to the Hospitals/Residents Problem, which bears some resemblance to the Teacher Relocation Problem studied here, is presented in [10]. In the Hospitals/Residents Problem, each resident is paired with an acceptable hospital, in such a way that a hospital's capacity is never exceeded. Both the hospitals and residents have preferences. Authors investigate four different techniques, two of them being similar to our assignment of binary values to variables.

VI. CONCLUSION AND FUTURE RESEARCH DIRECTIONS

We defined the teacher relocation problem, modeled it using constraints, implemented a solution as an ECL^{i}PS^e program, and tested our solution under various scenarios. We devised several strategies for reducing the search space, and generating results faster. The first is using a baseline value for pruning solution paths that cannot possibly improve the baseline value. Second is the usage of "incremental domains," where preferences of teachers are considered incrementally. Our simulation runs confirm that the incremental domain approach outperforms the non-incremental one.

The proposed model works well up to a reasonable number of teachers and quotas in schools. As the problem size becomes large (i.e. when the number of teachers increase and/or the schools quotas become small), since the search space grows exponentially, it may not be possible to find the best solution in a reasonable amount of time. However, North Cyprus is a small country, and the problem sizes seem well within the capability of our solution.

For future work, we plan to improve the "cost function" by taking into account the needs of the schools and how long a teacher has been waiting for relocation, handle the case of new teachers who are just beginning their careers, as well as teachers who will leave the system through retirement. A Web based user interface is also being planned.

REFERENCES

[1] F. Rossi, "Constraint Logic Programming," in *Proceedings of ERCIM/Compulog Net workshop on constraints*, Springer, LNAI 1865, 2000.

[2] IC-PARC. ECLiPSe 5.7 User Manual, 2003.

[3] J. Paralic, J. Csonto, M. Schmotzer, "Optimal Scheduling Using Constraint Logic Programming," in *Proceedings of 8th Symposium on Information Systems IS'97*, Varazdin, October 1997, pp. 65-72.

[4] S. Abdennadher, H. Schlenker, "Nurse Scheduling using Constraint Logic Programming," in *Proceedings of Eleventh Annual Conference on Innovative Applications of Artificial Intelligence, IAAI-99*, The MIT Press, Orlando, Florida, July 1999.

[5] G. Weil, K. Heus, P. Francois, M. Poujade, "Constraint programming for nurse scheduling," in *Proceedings of IEEE*

Engineering in Medicine and Biology, July/August 1995, pp. 417-422.

[6] M. Kambi, D. Gilbert, "Timetabling in Constraint Logic Programming", in *Proceedings of INAP-96: Symposium* and *Exhibition on Industrial Applications of Prolog*, Tokyo, Japan, 1996.

[7] K. Marriot, P. Stuckey, *Programming with Constraints: An introduction*, The MIT Press, 1998.

[8] J. Csonto, J. Paralic, "The CLP Approach to Solve a Scheduling Application," in *Proceedings of the 7th International Symposium on INFORMATION SYSTEMS'96*, Varazdin, September 1996, pp. 179-190.

[9] A. Schaerf, "Scheduling Sport Tournaments using Constraint Logic Programming," in *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI*-96), Budapest, Hungary, 1996, pp 634-639.

[10] D.F. Manlove, G. O'Malley, P. Prosser, C. Unsworth, "A Constraint Programming Approach to the Hospitals / Residents Problem," in *Proceedings of the Fourth Workshop on Modeling and Reformulating Constraint Satisfaction Problems*, *held at the 11th International Conference on Principles and Practice of Constraint Programming (CP 2005),* pp 28-43.