Antoon Kolen

A genetic algorithm for the partial binary constraint satisfaction problem: an application to a frequency assignment problem

RM/06/045

JEL code: C61

Maastricht research school of **E**conomics
of **TE**chnology and **OR**ganizations

Universiteit Maastricht
Faculty of Economics and Business Administration
P.O. Box 616
NL - 6200 MD Maastricht

phone : ++31 43 388 3830
fax : ++31 43 388 4873

A genetic algorithm for the partial binary constraint satisfaction problem: an application to a frequency assignment problem.

Antoon Kolen *

October 12, 2006

Abstract

We describe a genetic algorithm for the partial constraint satisfaction problem. The typical elements of a genetic algorithm, selection, mutation and cross-over, are filled in with combinatorial ideas. For instance, cross-over of two solutions is performed by taking the one or two domain elements in the solutions of each of the variables as the complete domain of the variable. Then a branch-and-bound method is used for solving this small instance. When tested on a class of frequency assignment problems this genetic algorithm produced the best known solutions for all test problems. This feeds the idea that combinatorial ideas may well be useful in genetic algorithms.

1 Introduction

A constraint satisfaction problem (CSP) is a problem defined by a finite set of variables, each of which has a finite set of possible values (the domain). Next, a set of constraints defined on these variables is given, which determine feasible combinations of domain elements of multiple variables. We consider only binary constraints, where the number of variables involved in each constraint is restricted to two. So, a constraint may forbid only pairs of domain elements. A solution of a binary constraint satisfaction problem consists of exactly one domain value in the domain of each variable, such that no forbidden combinations are present. We add one more feature to the binary constraint satisfaction problem (BCSP), namely penalties for domain values in the domain elements and penalties for pairs of domain elements. The value of a solution is the sum of the penalties. The objective of this partial binary constraint satisfaction problem (PBCSP) is to find a solution of minimum value. Note that standard constraints can be modeled with penalties by incurring very high penalties for forbidden combinations. Therefore, in the sequel we will only consider constraints with penalties.
In [3] genetic algorithms have been described for CSP problems. We develop a genetic algorithm specifically suitable for PBCSP. The main feature of this algorithm is the procedure for cross-over. Here two solutions, the parents, are selected from a population. The one or two values of each variable in these solutions are taken as the domain of the variable. Then, an exact algorithm will solve this small PBCSP to optimality. Note that

*Antoon Kolen deceased on October 3, 2004. This manuscript has been updated by Stan van Hoesel, Faculty of Economics and Business Administration, University Maastricht, P.O. Box 616, 6200 MD Maastricht, The Netherlands, Email: s.vanhoesel@ke.unimaas.nl

this means that the offspring solution is at least as good as each of the parents. Thus, if each of the solutions in the parent population is selected at least once, the offspring population is at least as good as the parent population. The genetic algorithm is tested on a set of problems, arising in frequency assignment.
In section 2 we will formally describe the PBCSP, and give examples. In Section 3 we describe the genetic algorithm for the partial constraint satisfaction problem, where as in Section 4 we solve the cross-over problem, and in Section 5 we apply the genetic algorithm to the RLFAP and discuss computational results for the CALMA -instances.

2 The PBCSP and examples: Frequency Assignment and MAX-SAT

Formally, a PBCSP can be described with the following tuple: $(G = (V, E), D_V, Q_V, P_E)$. Here,

1. $G = (V, E)$, an undirected graph called the *constraint graph*, the vertices correspond to the *variables*, and the edges correspond to the *constraints.*
2. $D_V = \{D_v | v \in V\}$, where D_v is a finite set, called the *domain* of variable $v \in V$,
3. $Q_V = \{Q_v : D_v \mapsto \mathbb{R} \mid v \in V\}$, where Q_v defines a *variable penalty* function which assigns a real value to every domain value in D_v, $v \in V$,
4. $P_E = \{P_{\{v,w\}} : D_v \times D_w \mapsto \mathbb{R} \mid \{v, w\} \in E\}$, where $P_{\{v,w\}}$ defines a *constraint penalty* function which assigns a real value to every pair of domain elements in $D_v \times D_w$, $\{v, w\} \in E$.

A *solution* is defined to be a vector $(d_v)_{v\in V}$ containing exactly one domain element $d_v \in D_v$ for every vertex $v \in V$. The *value* of a solution $(d_v)_{v\in V}$ is defined to be the sum of all vertex and edge penalties, i.e., $\sum_{v\in V} Q_v(d_v) + \sum_{\{v,w\}\in E} P_{\{v,w\}}(\{d_v, d_w\})$. The objective is to find a solution of minimum value.
The PBCSP can be formulated as a {0,1}-programming problem. Let us define for all $v \in V$ and all $d_v \in D_v$ the $\{0, 1\}$-variables

$$y(v, d_v) = \begin{cases} 1 & \text{if } d_v \in D_v \text{ is selected} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

and for all $\{v, w\} \in E, d_v \in D_v$ and $d_w \in D_w$ the $\{0, 1\}$-variables

$$x(v, d_v, w, d_w) = \begin{cases} 1 & \text{if } \{d_v, d_w\} \in D_v \times D_w \text{ is selected} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Furthermore, let us define for all $v \in V$ and for all $d_v \in D_v$ $q(v, d_v) = Q_v(d_v)$ and for all $\{v, w\} \in E$, for all $d_v \in D_v$ and for all $d_w \in D_w$ $p(v, d_v, w, d_w) = P_{\{v,w\}}(\{d_v, d_w\})$.
Then PBCSP can be formulated as a

$$\min : \sum_{v \in V} \sum_{d_v \in D_v} q(v, d_v) y(v, d_v) + \sum_{\{v,w\} \in E} \sum_{d_v \in D_v} \sum_{d_w \in D_w} p(v, d_v, w, d_w) x(v, d_v, w, d_w)$$

s.t.

$$\begin{aligned}
&\sum_{d_v \in D_v} y(v, d_v) = 1 && \forall_{v \in V} \\
&\sum_{d_w \in D_w} x(v, d_v, w, d_w) = y(v, d_v) && \forall_{\{v,w\} \in E} \ \forall_{d_v \in D_v} \\
&y(v, d_v) \in \{0, 1\} && \forall_{v \in V} \ \forall_{d_v \in D_v} \\
&x(v, d_v, w, d_w) \in \{0, 1\} && \forall_{\{v,w\} \in E} \ \forall_{d_v \in D_v} \ \forall_{d_w \in D_w}
\end{aligned} \tag{3}$$

The first set of constraints expresses that exactly one value in the domain of every variable must be selected. The next set of constraints expresses the following: if d_v is selected then for exactly one combination $d_v \ d_w \in D_w$ is selected. If d_v is not selected, then also none of the combinations $d_v \ d_w \in D_w$ is selected.

2.1 Frequency assignment

A straightforward example of a PBCSP is the radio link frequency assignment problem studied in the CALMA-project. In the CALMA (**C**ombinatorial **AL**gorithms for **M**ilitary **A**pplications)-project researchers from England, France, and the Netherlands tested different combinatorial algorithms on a set of 11 frequency assignment problems. The genetic algorithm to be described in this paper found the best known solutions to all test problems and for some test problems outperforms other solution techniques. Results of the CALMA-project as well as all test problems are described in [2] and available by anonymous ftp from ftp.win.tue.nl in the directory /pub/techreports/CALMA.

The radio link frequency assignment problem (RLFAP) is defined by a set of radio links partitioned into pairs, where each pair consists of a receiver and a transmitter radio link. Associated with a radio link is a set of frequencies which can be assigned to it. The distance between the frequencies assigned to the receiver and transmitter must be equal to a fixed distance. For some pairs of radio links there is an interference constraint which states that the distance between the two assigned frequencies should be greater than a given required distance. It is a soft constraint because it may be violated at a certain interference cost. Both the required distance and the associated cost depend on the two radio links involved. For some receiver-transmitter pairs a preferred pair of frequencies is given. It is possible to deviate from these preferred frequencies at a certain mobility cost which depends on the receiver-transmitter pair. The objective is to find a frequency assignment that minimizes total cost, i.e., the sum of total interference cost and total mobility cost. An overview on models and algorithms for frequency assignment can be found in [1].

To model RLFAP as a PBCSP we introduce a variable for every receiver-transmitter pair. The domain of a variable consists of every pair of frequencies, one from the domain of the receiver radio link and one from the transmitter radio link, which satisfy the fixed distance requirement. The variable penalty of a pair of frequencies is equal to the mobility cost. There is a constraint for two receiver-transmitter pairs when there is at least one interference constraint between two radio links one from each receiver-transmitter pair.

The constraint penalty for a pair is equal to the sum of the interference cost of the four involved radio links (two receivers-transmitter pairs). For two receiver-transmitter pairs the mutual distance requirements and interference cost are given below.

	r_2	t_2
r_1	351	518
t_1	201	351

distances

	r_2	t_2
r_1	1000	10
t_1	1	1000

costs

For the CALMA instance the distance between the frequencies assigned to a receiver pair must be equal to 238. If we assign (72,310) to (r_1,t_1) and (414,652) to (r_2,t_2), then according to the tables above the corresponding constraint penalty is 1000 (for r_1 and r_2) +1 (for t_1 and r_2) = 1001.

2.2 MAX-SAT

The maximum satisfiability problem (MAX-SAT) can be modeled as a PCSP. Let C_1,C_2, ..., C_m be a set of clauses defined on the boolean variables $x_1, x_2, .., x_n$. Each clause is a disjunction of literals, where a literal is either a boolean variable x_i or its negation $\overline{x}_i$. The objective is to assign to each boolean variable the value true or false so as to satisfy the maximum number of clauses or equivalently minimize the number of clauses that are not satisfied. A clause is satisfied if and only if at least one literal in the clause has the value true. An example of the maximum satisfiability problem is given by

$$(x_1 \vee x_3) \quad \wedge \quad (x_1 \vee x_2 \vee x_3) \quad \wedge \quad (x_2 \vee \overline{x}_3)$$

Example 1.

$$C_1 \qquad\qquad C_2 \qquad\qquad C_3$$

To model the maximum satisfiability problem as a PBCSP we introduce a variable for each clause and a variable for each boolean variable. The domain of a clause variable contains the literals in the clause. We denote a domain element by the index of corresponding literal. The domain of a variable corresponding to a boolean variable consists of the values $true$ and $false$. There is a constraint between a variable corresponding to a clause and a variable corresponding to a boolean variable whenever the boolean variable or its negation occurs in the clause.

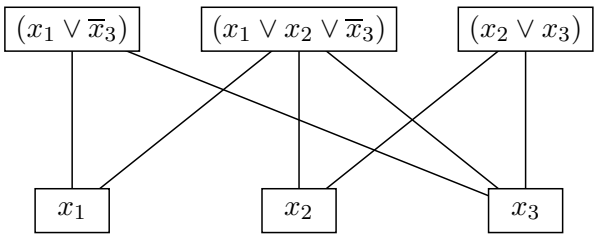

Figure 1: Constraints in example MAX-SAT instance.

The constraint penalty of any pair of conflicting domain values is equal to one; all other constraint penalties are zero. So, the domain pair $\{x_i, false\}$ is conflicting, and thus incurs a penalty of one. The same holds for the pair $\{\overline{x}_i, true\}$. There are no other penalties. Figure 2 shows the variables and constraints corresponding to Example 1 where conflicting pairs of domain values are indicated by a line.

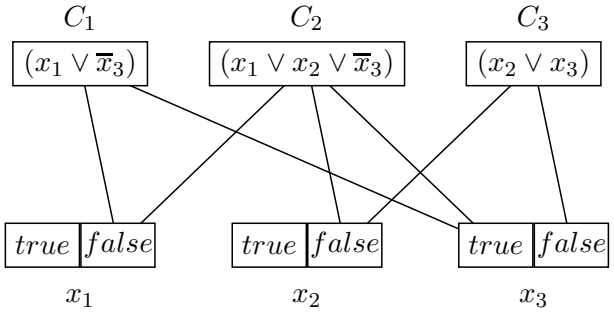

Figure 2: Penalties in example MAX-SAT instance.

Theorem 1. *Consider an instance of MAX-SAT with n variables and m clauses. Then, MAX-SAT has value k if and only if the corresponding instance of PBCSP has value $m-k$.*

Proof. Consider a truth assignment of the boolean variables of the maximum satisfiability problem that satisfies k of the m clauses. In the corresponding partial binary constraint satisfaction problem we select the domain values of the variable corresponding to boolean variables as defined by the truth assignment. For the k clauses that are satisfied we select a literal that has the value true. For the $m-k$ clauses that are not satisfied we select an arbitrary literal. Then the value of this solution is $m-k$. This proves that, if the optimal value of the maximum satisfiability problem is k, then the optimal value of the partial binary constraint satisfaction problem is at most $m-k$.
Consider an optimal solution of the partial constraint problem with value $m-k$. Then there are exactly $m-k$. clauses for which the selected literal is involved in a conflict and since the solution is optimal any other literal selected in the same clause would result in a conflict as well. Hence the domain values of the variables corresponding to boolean variables define a truth assignment that satisfies exactly k clauses.We conclude that if the optimal value of the partial binary constraint problem is $m-k$, then the optimal value of the maximum satisfiability problem is at least k. □

If for Example 1 we consider the truth assignment $x_1 = true$, $x_2 = false$, $x_3 = false$, then C_1 and C_2 are satisfied, and C_3 is violated. For the partial binary constraint satisfaction problem we select $true$ for x_1, $false$ for x_2, $false$ for x_3, x_1 for C_1, x_1 for C_2, and x_2 for C_3. If in Figure 1 we select x_1 for C_1, x_1 for C_2, x_2 for C_3, $true$ for x_1,$true$ for x_2, and $false$ for x_3, then we have the optimal solution of the partial binary constraint satisfaction problem of value zero. The corresponding truth assignment $x_1 = true$, $x_2 = true$, and $x_3 = false$ satisfies all clauses.
We conclude from the above results that the optimal value of the partial binary constraint satisfaction problem is $m-k$ if and only if the optimal value of the maximum satisfiability problem is k.

3 The genetic algorithm.

Local search algorithms, such as simulated annealing and tabu search, construct a sequence of solutions, where the successor of a given solution in the sequence is a so-called neighbor of that solution. The definition of the neighborhood of a given solution is a basic ingredient of any local search algorithm.

A genetic algorithm constructs a sequence of populations of solutions, where the successor of a given population in the sequence is a so-called neighbor of that population. A population consists of a set of solutions. The basic ingredients of a genetic algorithm are selection, cross-over, and mutation procedures. A population is called a neighbor of a given population if it can be obtained from that population by applying these procedures to the given population. A new population is constructed from an old population by repeatedly applying cross-over to pairs of elements of the old population selected by the selection procedure. The idea of cross-over is to use the genetic material of the two input elements to construct two elements of the new population, which are hopefully better than the two input elements. The mutation procedure changes a given solution. For a more detailed discussion of genetic algorithms we refer to the book by Goldberg (1988). No previous knowledge of genetic algorithms is required to understand the description of our genetic algorithm.

Let us call two solutions *neighbors* if they differ in exactly one component, i.e., one domain element. A *1-optimal* solution is a solution with the property that no neighbor exists with a smaller value. An element of our genetic algorithm is defined to be a 1-optimal solution. A population of our genetic algorithm consists of a fixed number (called the population size) of (not necessarily distinct) elements. The initial population of our genetic algorithm is constructed by generating random solutions, and by applying a local optimization algorithm to each solution to make it 1-optimal. Our genetic algorithm consists of a number of generations. Each generation starts with a given population, called the parent population, and constructs a new population of the same size, the offspring. This offspring population is the parent population of the next generation. The offspring population will have the property that it is at least as good as the parent population in the sense that every element of the parent population is replaced by an element with a value which is less than or equal to the value of the element it replaces. The genetic algorithm returns the element of minimum value in the last population.

To construct a parent population from the offspring population we randomly select for each element of the old population a partner element from the offspring population. This partner element is selected with a probability which is proportional to the reciprocal of its value. This means that elements with a small value are preferred in the selection procedure. Next we apply the cross-over function to the given element and the random selected partner element. The cross-over function takes as input two elements (called parents) and returns an element (called offspring) with a value that is less than or equal to the value of both input elements. So, in this cross-over only one offspring is created, in contrast with the two offspring in traditional cross-over. After applying a local optimization procedure (the mutation procedure) to make this offspring 1-optimal it will replace the given element in the parent population. The construction of the parent population from the offspring population is summarized below.

```
for i:=1 to PopSize do
{
    (Selection)
    Select ParentPopulation[i] and random select ParentPopulation[j] with a
    probability that is inverse proportional to its objective value.

    (Cross-over)
    OffspringPopulation[i] := CrossOver(ParentPopulation[i],ParentPopulation[j]);
```

(Mutation)
Local optimize OffspringPopulation[i] to make it 1-optimal;
}
The idea of the cross-over is that each domain value of the offspring will be inherited from one of its parents. Given two parent solutions we can consider the restricted PBCSP where the domain of a variable contains the set of domain values assigned to it in one of the two parents. So the domain of a variable contains exactly one domain value if it is assigned the same domain value in both parents, otherwise the domain consists of the two different domain values assigned to it in both parents. The offspring of our cross-over procedure will be the optimal solution of this restricted partial constraint satisfaction problem. Since both parents are solutions of this problem, the optimal solution, i.e., the offspring, will have a value which is less than or equal to the minimum value of its parents. We will discuss an exact solution method of the restricted PBCSP in Section 4.
Our genetic algorithm as described above differs from more traditional genetic algorithms in two aspects. In a traditional genetic algorithm a cross-over produces two offsprings instead of the one offspring as in our cross-over. In a traditional genetic algorithm both parents to which the cross-over is applied are selected randomly. The probability of being selected is determined by a so-called fitness function. In our genetic algorithm the fitness of an element is defined as the reciprocal of its value. Although we have experimented with more traditional genetic algorithms using our cross-over the best results were obtained using the version as described above. An intuitive explanation for the success of our algorithm may be that every element of the population is involved in a cross-over at least once, while good solutions have a high probability of being involved in more than one cross-over. So although we prefer good genetic material (i.e., domain values) we do not exclude any genetic material beforehand. The local optimization procedure we apply to the offspring of the cross-over procedure to make it 1-optimal is used as the mutation operator of our genetic algorithm.

4 The cross-over

Consider the {0,1}-programming formulation of PBCSP of section 2. Before solving this problem we first apply some preprocessing techniques to reduce the size of the problem. During the preprocessing we will always make sure that for every edge $\{v, w\} \in E$ with $D_v = \{a_v, b_v\}$ and $D_w = \{a_w, b_w\}$ the edge penalties satisfy $\min\{p(v, a_v, w, a_w), p(v, a_v, w, b_w)\} = 0$, $\min\{p(v, b_v, w, a_w), p(v, b_v, w, b_w)\} = 0$, $\min\{p(v, a_v, w, a_w), p(v, b_v, w, a_w)\} = 0$, and $\min\{p(v, a_v, w, b_w), p(v, b_v, w, b_w)\} = 0$.
Assume that these conditions do not hold, say $\min\{p(v, a_v, w, a_w), p(v, a_v, w, b_w)\} > 0$. Then we can increase the vertex penalty $q(v, a_v)$ with $\min\{p(v, a_v, w, a_w), p(v, a_v, w, b_w)\}$ and decrease both $p(v, a_v, w, a_w)$ and $p(v, a_v, w, b_w)$ with $\min\{p(v, a_v, w, a_w), p(v, a_v, w, b_w)\}$ without affecting the total penalty of a solution.
We use five different types of preprocessing techniques, namely deleting constraints, deleting variables with a domain of cardinality one, deleting variables of degree one in the constraint graph, deleting variables of degree two in the constraint graph, and deleting dominated domain values. Although we will discuss these preprocessing techniques in the context of our cross-over problem, they apply to any partial binary constraint satisfaction problem.

1. A constraint is deleted whenever all constraint penalties are equal

2. Consider a variable $v \in V$ with a domain of cardinality one, say $D_v = \{a_v\}$.

   For an edge $\{v, w\} \in E$ the edge penalty only depends on which d_w is selected from D_w. Therefore the edge penalty $p(v, a_v, w, d_w)$ can be viewed as a vertex penalty for d_w. If we define $q(w, d_w) := q(w, d_w) + p(v, a_v, w, d_w)$ for all $\{v, w\} \in E$ and for all $d_w \in D_w$, and delete vertex v and all edges incident to it from the constraint graph, then there is a one-to-one correspondence between solutions of the reduced problem and solutions of the original problem. The objective value of a solution to the original problem is equal to the objective value of the corresponding problem of the reduced problem plus $q(v, a_v)$.

3. Consider a variable $v \in V$ of degree one in the constraint graph, and let $\{v, w\}$ the edge incident to v.

   If $d_w \in D_w$ is assigned to w, then the domain value $d_v^* \in D_v$ assigned to v will be selected so as to minimize the sum of the vertex penalty of v and the edge penalty of $\{v, w\}$, i.e., $q(v, d_v^*) + p(v, d_v^*, w, d_w) = \min\{q(v, d_v) + p(v, d_v, w, d_w) | d_v \in D_v\}$. If we define $q(w, d_w) := q(w, d_w) + \min\{q(v, d_v) + p(v, d_v, w, d_w) | d_v \in D_v\}$, and delete vertex v and the edge $\{v, w\}$, then the optimal value of the reduced and original problem are the same. Given an optimal solution of the reduced problem with d_w assigned to w we can find an optimal solution to the original problem by assigning the domain value d_v^* defined above to v.

4. Consider a variable $v \in V$ of degree two in the constraint graph, and let $\{u, v\}$ and $\{v, w\}$ the edges incident to v.

   If $d_u \in D_u$ is assigned to u and $d_w \in D_w$ is assigned to w, then the domain value $d_v^* \in D_v$ assigned to v will be selected so as to minimize the sum of the vertex penalty of v and the edge penalties of $\{u, v\}$ and $\{v, w\}$, i.e, $p(u, d_u, v, d_v^*) + q(v, d_v^*) + p(v, d_v^*, w, d_w) = \min\{p(u, d_u, v, d_v) + q(v, d_v) + p(v, d_v, w, d_w) | d_v \in D_v\}$. If there is no edge $\{u, w\}$ then we add the edge $\{u, w\}$ with all edge penalties equal to zero to the constraint graph. Clearly this does not affect the value of any solution. If we now delete the edges $\{u, v\}$ and $\{v, w\}$ from the constraint graph and define $p(u, d_u, w, d_w) := p(u, d_u, w, d_w) + \min\{p(u, d_u, v, d_v) + q(v, d_v) + p(v, d_v, w, d_w) | d_v \in D_v\}$, then the optimal value of the reduced and original problem are the same. Given an optimal solution of the reduced problem with d_u assigned to u and d_w assigned to w we can find an optimal solution to the original problem by assigning the domain value d_v^* defined above to v.

5. Our last preprocessing technique deals with dominated domain values.

   For a variable $v \in V$ we say that domain value $b_v \in D_v$ *dominates* domain value $a_v \in D_v$ whenever for every solution in which a_v is assigned to v we can find a solution in which b_v is assigned to v, and for which the latter solution has a value that is less than or equal to the value of the former solution.

   If b_v dominates a_v, then a_v can be deleted from the domain D_v of vertex v. If the original domain D_v has cardinality two, then the reduced domain has cardinality one and vertex v can be deleted using preprocessing technique 2.

We will formulate a sufficient condition for dominance.
If $\{v, w\}$ is an edge, then the worst increase in the edge penalty for this edge if a_v in a solution is replaced by b_v is given by $c(v, w) = \max\{p(v, b_v, w, d_w) - p(v, a_v, w, d_w) | d_w \in$

$D_w\}$. If $q(v, b_v) + \sum_{\{v,w\}\in E} c(v, w) \leq q(v, a_v)$, then b_v dominates a_v because replacing a_v by b_v in any solution will result in a solution which is at least as good. For any variable with a domain of cardinality two we check whether one domain value is dominated by the other domain value by checking the sufficient condition described above. If a domain value is dominated, then it is deleted from the domain.
After application of all preprocessing techniques the {0,1}-programming problem of the remaining partial binary constraint satisfaction problem is solved by a partial cutting plane algorithm. The class of valid inequalities that we use in the cutting plane algorithm is one of the classes defined in [4]. The inequalities are called 3-cycle inequalities.
Given vertices u, v, and w which form a 3-cycle in the constraint graph. Let the domains be given by $D_u = \{a_u, b_u\}, D_v = \{a_v, b_v\}$, and $D_w = \{a_w, b_w\}$. Then it is easy to see that the following 3-cycle inequality is a valid inequality

$$x(u, a_u, v, b_v) + x(v, a_v, w, b_w) + x(w, a_w, u, b_u) <= 1.$$

For a given 3-cycle there are four 3-cycle inequalities of this type which are facet defining for the convex hull of solutions of the partial binary satisfaction problem.. We start the solution procedure by first solving the linear programming relaxation. Then we add all 3-cycle inequalities that are violated by the current optimal linear programming solutions and re-optimize. This is repeated until all 3-cycle inequalities are satisfied. In almost all problems we solved we end with an integer optimal solution at this stage. If the optimal linear programming solution is fractional, then we solve the original {0,1}–programming problem with all added 3-cycle inequalities using CPlex 4.0 as mixed integer optimizer.

5 Computational results.

In the CALMA project two sets of problem instances were used. The CELAR instances are provided by the French ministry of defense. The GRAPH instances are random instances, generated by a group of researchers from Delft University of Technology, which have the same characteristics as the CELAR instances.
There were four problem instances (CELAR 9, CELAR 10, GRAPH 7, and GRAPH 12) which involved mobility penalties. These problem instances turned out to be very easy, and all tested solution techniques found the same answers (CELAR 9: 15571, CELAR 10: 31516, GRAPH 7: 4324, GRAPH 12: 11827). The genetic algorithm already found these solutions within 5 generations with a population size of 10. Therefore these problem instances are left out of the computational tests.
In Table 1 we list for each problem instance of our problem type its name as referred to in the CALMA project, the number of vertices ($|V|$) and the number of edges ($|E|$), the number of y-variables, which can be calculated from $\sum_{v\in V} |D_v|/|V|$, the number of x-variables, which can be calculated from $\sum_{\{v,w\}\in e} |D_v| * |D_w|/|E|$, and the best known solution value (O) for each problem. The best known solutions are obtained by the genetic algorithm described in the previous section.
Our genetic algorithm was implemented in C++ and run on a workstation (DEC 2100 A500MP). All linear and mixed integer program are solved using CPLEX 4.0. Although the computational time of our genetic algorithm already is very impressive, the code for the cross-over function is not yet fully optimized. Since the cross-over function is called many times, there is still room for improvement.

Name	$\|V\|$	$\|E\|$	$\sum_{v \in V} \|D_v\| \ / \ \|V\|$	$\sum_{\{v,w\} \in E} (\|D_v\|\|D_w\|) \ / \ \|E\|$	O
CELAR 6	100	350	40.10	1622.58	3389
CELAR 7	200	817	39.88	1619.43	343592
CELAR 8	458	1655	39.52	1510.91	262
GRAPH 5	100	416	37.08	1372.40	221
GRAPH 6	200	843	37.71	1416.48	4115
GRAPH 11	340	1425	37.70	1417.52	3080
GRAPH 13	458	1877	38.40	1481.52	10110

Table 1: Statistics problem instances.

Pop Size	Best Value	Worst Value	Mean Value
10	3739	5506	4359.3
40	3404	3575	3500.1
70	3391	3554	3449.5
100	3389	3407	3392.5
150	3389	3404	3391.9

Table 2: Results CELAR 6 for 10 runs.

Extensive computational experiments have shown that the best solution found by the genetic algorithm is always found within 10 generations for the CELAR instances and for the problem instance GRAPH 5, and within 15 generations for all other GRAPH problem instances.

Let us first demonstrate for problem instance CELAR 6 how the solution quality depends on the population size. For different population sizes we have generated 10 runs, where each run corresponds to applying the genetic algorithm for 10 generations. The best solution value, the worst solution value, and the mean solution value for these 10 runs are given in Table 2.

It can be seen from Table 2 that the genetic algorithm for CELAR 6 is very stable for population sizes greater than or equal to 100. Tables 3 and 4 show the results of the genetic algorithm with a population size of respectively 100 and 150 for all problem instances. We list the best value, the worst value, the mean value, the number of generations, and the average CPU time per run.

To demonstrate how the solution quality depends on the number of generations we have used the best run with population size 150 for every problem instance. In Table 5 we list the best solution value found after every generation. Note that the best solution value for generation 0 corresponds to the best 1-optimal solution in the initial population.

We can conclude from these computational results that for population sizes of at least 100 the genetic algorithm is very robust, i.e., there is only a small variance in the objective value of the solution produced in each run.

Since no optimal solutions or good lower bounds are known for these problems we can only compare the results with those obtained by other groups in the CALMA project. The genetic algorithm produced the best known solutions to all problem instances. For some problem instances (GRAPH 11 and GRAPH 13) the best objective value found is

Name	Best Value	Worst Value	Mean Value	Nr Gen	CPU min
CELAR 6	3389	3407	3392.5	10	3
CELAR 7	343593	353693	344784	10	9
CELAR 8	262	263	262.1	10	24
GRAPH 5	221	269	240.4	10	3
GRAPH 6	4138	5145	4431.1	15	5
GRAPH 11	3126	4045	3468.2	15	19
GRAPH 13	10234	11012	10486.2	15	36

Table 3: Results for 10 runs with population size 100.

Name	Best Value	Worst Value	Mean Value	Nr Gen	CPU min
CELAR 6	3389	3404	3391.9	10	5
CELAR 7	343592	343794	343643	10	13
CELAR 8	262	263	262.2	10	36
GRAPH 5	221	236	232.5	10	4
GRAPH 6	4125	4166	4130.7	15	7
GRAPH 11	3088	3138	3109.5	15	29
GRAPH 13	10110	10350	10223.1	15	55

Table 4: Results for 10 runs with population size 150.

Gen	C6	C7	C8	GR5	GR6	GR11	GR13
0	8021	10272611	944	6223	14814	32106	50999
1	5686	3911098	527	2927	9997	25185	36809
2	4062	748798	392	1049	8104	15186	31115
3	3710	465445	299	427	6667	11221	24936
4	3430	375109	278	263	5699	6812	20818
5	3423	353911	270	246	5251	4785	15985
6	3392	344603	266	231	4879	4100	13928
7	3391	343697	264	231	4434	3677	11736
8	3389	343697	262	224	4187	3422	10829
9	3389	343592	262	221	4132	3298	10459
10	3389	343592	262	221	4130	3249	10240
11					4127	3208	10239
12					4127	3129	10152
13					4127	3124	10110
14					4125	3223	10110
15					4125	3088	10110

Table 5: Best solution value per generation.

10% better than the best known solution found by another technique. It still remains a challenge to solve these problem instances to optimality.

References

[1] K. I. Aardal, C. P. M. van Hoesel, A. M. C. A. Koster, C. Mannino, and A. Sassano. Models and solution techniques for the frequency assignment problem. *4OR*, 1(4):261–371, 2003.

[2] K. I. Aardal, C. A. J. Hurkens, J. K. Lenstra, and S. R. Tiourine. Algorithms for radio link frequency assignment: The CALMA project. *Operations Research*, 50(6):968–980, 2002.

[3] D. Goldberg. *Genetic algorithms in search, optimization & machine learning*. Addison Wesley, 1989.

[4] A. M. C. A. Koster, C. P. M. van Hoesel, and A. W. J. Kolen. The partial constraint satisfaction problem: Facets and lifting theorems. *Operations Research Letters*, 23(3–5):89–97, 1998.