

Tackling Car Sequencing Problems Using a Generic Genetic Algorithm

Terry Warwick & Edward P K Tsang

Department of Computer Science

University of Essex

Colchester CO4 3SQ

United Kingdom

email: {warwt, edward}@essex.ac.uk

A revised version to appear in Evolutionary Computation, MIT Press

Key words: genetic algorithms, constraint satisfaction, car sequencing

Abstract

The car sequencing problem (CarSP) was seen as a challenge to artificial intelligence. The CarSP is a version of the *job-shop scheduling problem* which is known to be NP-complete. The task in the CarSP is to schedule a given number of cars (of different types) in a sequence to allow the teams in each work station on the assembly line to fit the required options (e.g. radio, sunroof) on the cars within the capacity of that work station. In unsolvable problems, one would like to minimize the penalties associated to the violation of the capacity constraints. Previous attempts to tackle the problem have either been unsuccessful or restricted to solvable CarSPs only. In this paper, we report on promising results in applying a generic *genetic algorithm*, which we call GAcSP, to tackle both solvable and unsolvable CarSPs.

1 Introduction

1.1 Overview

Constraint satisfaction is a general problem which is found in many areas. A *constraint satisfaction problem* (CSP) is a problem in which one would like to assign a value to a set of variables, satisfying a set of constraints (the CSP will be defined more formally later). The generality and importance of constraint satisfaction has led to active research in this field in recent years and the development of commercial constraint problem solvers, such as CHIP and ILOG Solver (Cras, 1993). One of the areas in which success has been reported is scheduling. In industrial scheduling, resources are typically

scarce, and therefore, many CSPs have no solution. A *Partial constraint satisfaction problem* (PCSP) is a problem in which constraints may be violated at certain pre-defined costs (Freuder & Wallace, 1992) (Wallace & Freuder, 1993).

Typical algorithms for solving PCSPs are variants of branch-and-bound, which application is limited by the combinatorial explosion problem because of the NP-completeness nature of PCSPs. In Freuder *et. al.* 1995), Tsang argued for the role of genetic algorithms in partial constraint satisfaction. In this paper, we present a generic *genetic algorithm* (GA) strategy which we call GAcSP, which is designed to tackle PCSPs. GAcSP is a combination of a GA with repair and hill-climbing. GAcSP has been demonstrated to be successful in another PCSP, namely, the processor configuration problem (Warwick & Tsang, 1993). In this paper, we describe its application to the *car sequencing problem* (CarSP), which is a version of the *job-shop scheduling problem*. The CarSP is known to be NP-complete. It was seen as a challenge to *artificial intelligence* (AI) (Parrello, 1988).

PCSP and GA are summarized in Sections 1.2 and 1.3. The objective of this research is described in Section 1.4. The CarSP is described in detail and formalized as a PCSP in Section 2. GAcSP is described in Section 3. In Section 4 we present empirical results on testing GAcSP on both solvable and unsolvable CarSPs. We compare the performance of GAcSP with other heuristic techniques which are applicable to solvable as well as unsolvable PCSPs. Section 5 concludes the paper.

1.2 Partial Constraint Satisfaction Problems

The constraint satisfaction problem (CSP) is an important class of problems in AI and computer science (Tsang, 1993; Freuder & Mackworth, 1994). Instances of CSPs include scheduling, scene labeling, graph isomorphism, boolean satisfiability and graph coloring. The CSP comprises a finite set of variables, each of which has a finite domain, and a finite set of constraints. A solution tuple is an assignment of a value to each variable (from their respective domains) satisfying the constraints. Following (Tsang 1993), we formally define a CSP as follows:

Definition 1 (CSP): A constraint satisfaction problem (CSP) is a triple:

$$(Z, D, C),$$

where $Z = \text{set of variables } \{x_1, x_2, \dots, x_N\}$;

$D = \text{a function which maps every variable } x_i \text{ to a discrete domain } D_i$;

and $C = \text{a set of constraints on an arbitrary subset of variables in } Z, \text{ restricting the values that they can take simultaneously.}$

In other words, each discrete variable x_i has a domain $D_i = \{v_{i1}, v_{i2}, \dots, v_{ik}\}$ where cardinality $k = |D_i|$.

A *label* is an assignment of a value to a variable. A *compound label* is a set of labels for a set of variables.

The PCSP is an optimization problem, where an objective function g is defined which maps every compound label to a numeric value. The task is to find a compound label with the optimal (which can be maximal or minimal, depending on the problem specification) value. In other words, PCSPs are CSPs where there may not be a solution which satisfies all the constraints, in which case the requirement is to find “the best” solution tuple which minimizes or maximizes the objective function. Later we shall show that the CarSP is an instance of a PCSP.

1.3 Genetic Algorithms

GAs are stochastic search techniques which explore combinatorial search spaces using simulated evolution (Holland, 1975). Exploration is achieved through the recombination of data structures (which represent candidate solutions) which are given a *fitness* value according to a domain specific objective function. Selection of data structures from a population based upon the relative fitness of the data structures exploits those that are more successful in minimizing or maximizing the objective function. GAs can converge to near optimal solutions, but generally lack a local improvement ability. In this research, we test a strategy named GAcSP which combines the GA robust search technique with a local improvement ability. We argue that such a combination provides an effective approach for tackling PCSPs.

1.4 Motivation and Objective

In some scheduling problems, such as resource allocation, we would like to optimize certain cost or utility. These problems are CSPs with the additional requirement of optimizing a domain specific objective function. PCSPs are difficult because they effectively require all solutions to be found and compared in order to find an optimal solution. PCSPs with tight constraints can be tackled by complete methods (e.g. branch-and-bound) where efficient heuristics can reduce the size of the space to be searched. On the other hand, PCSPs with loose constraints have a much larger proportion of the space to be searched and are therefore potentially more difficult. Methods for tackling PCSPs are faced with the difficulty of having to compare all solutions to find the one which violates the constraints of the least cost (should constraints need to be violated).

When complete search methods cannot be expected to obtain solutions to PCSPs within a reasonable time period because of the combinatorial explosion problem or lack of efficient heuristics, stochastic methods can be used. Stochastic methods such as GA, *Heuristic Repair* (HR) (Minton, et. al. 1990), GENET (Davenport *et. al.* 1994) or GSAT (Selman *et. al.* 1992, 1993) are incomplete search techniques which sacrifice completeness and settle for near optimality to be achieved in an acceptable period of time. GAs have been demonstrated successful on combinatorial optimization problems (such as TSP and QAP), and have shown promise when applied to constraint optimization problems (Tsang & Warwick, 1990) (Michalewicz *et. al.* 1989, 1991). Motivated by the generality and importance of PCSPs, the objective of this research is to develop a generic GA-based tool for tackling them efficiently.

2 The Car Sequencing Problem (CarSP)

2.1 Definitions

In a CarSP, one is given a set of pre-defined car types, each of which requires a different set of options (e.g. car radio, seat covers etc.) to fitted by specialized teams in workstations on an assembly line. The task is to sequence a specified number of cars for each car type so that workstation teams can fit the required options whilst the scheduled cars pass through the workstations. For k car types, there are

$pr[1], \dots, pr[k]$ *production requirements* of the CarSP. We can calculate the total number N of cars to be sequenced using Equation 1:

$$N = \sum_{j=1}^k pr[j] \quad (1)$$

The complexity of a CarSP is thus N^k . We define a set O of option requirements for an n option CarSP with k car types as: $O[m,j]$ for all $m = 1, 2, \dots, n$ and $j = 1, 2, \dots, k$. $O[m,j] = 1$ if option m is required by car type j ; 0 otherwise.

For each option m there is a specialist workstation on the assembly line with a team or teams of workers which can fit p_m options in the time it takes for q_m cars to pass through the workstation. This represents the *capacity constraint* $p_m : q_m$ on that option. We can calculate the number of option m required in a schedule:

$$O_{num}(m) = \sum_{j=1}^k (pr[j] \times O[m,j]) \quad (2)$$

Also, the maximum number of option m allowed in a schedule by the capacity constraint $p_m : q_m$ (for simplicity, we assume that N is divisible by q_m) is:

$$O_{max}(m) = \frac{p_m}{q_m} \times N \quad (3)$$

The level of resources utilization in the workstation for option m can be measured by the *utility ratio* for m as:

$$u_m = \frac{O_{num}(m)}{O_{max}(m)} \quad (4)$$

A necessary but not a sufficient condition for a CarSP to be solvable is that all capacity constraints $p_m : q_m$ must be satisfiable, that is $\forall m: (u_m \leq 1)$.

The overall level of resources utilization in a CarSP can be characterized by the *average utility*:

$$\mu = \frac{\sum_{m=1}^n u_m}{n} \quad (5)$$

We can demonstrate these ideas with a simple example CarSP presented in Table 1.

Table 1: Example of a solvable CarSP

option	car type				capacity constraint			
	1	2	3	4	$p:q$	O_{max}	O_{num}	u_m
1 car radio	1	0	0	1	1:2 = 0.50	6	5	0.83
2 furry dice	0	1	0	1	2:3 = 0.66	8	6	0.75
3 power steering	0	0	1	0	1:3 = 0.33	4	4	1.00
production requirement ($pr[i]$):	2	3	4	3			$\mu =$	0.86

In this problem, 12 cars of four types need to be produced, and three options are available. Car radio (option 1) needs to be fitted in cars of types 1 and 4. Two cars of type 1 and three cars of type 4 must be produced, and therefore $O_{num}(1) = 2 + 3 = 5$. The capacity constraint for car radio is 1:2, or 50%. A total of 12 cars need to be scheduled. Therefore, $O_{max}(1) = 50\% \times 12 = 6$. The following example schedule S satisfies the capacity constraints in Table 1:

position I in S : 12 11 10 9 8 7 6 5 4 3 2 1
car type (1 to m): 4 3 4 2 1 3 2 1 3 2 4 3 \rightarrow assembly line

In this schedule, position 1 is assigned to car type 3, position 2 to car type 4, etc. This schedule satisfies the capacity constraint of, say, car radio (which is named option 1). This is because $p_1:q_1$ is 1:2, and no two consecutive positions are assigned to car types 1 and 4 (which, according to table 1, are the only car types which require radios to be fitted).

2.2 The Penalty Function

There are CarSPs which are not solvable because the capacity constraints cannot be satisfied (i.e. $u_m > 1$). In these problems, *penalty functions* are used to minimize the capacity constraint violation and encourage spacing between options (Parrello *et. al.* 1986). Adding option 3 to type 1 cars will make the previous example CarSP unsolvable as in Table 2 (differences from Table 1 are in bold).

Table 3: Summary GAcSP, HR and TABU Experiment 4.2 Results

option	car type				capacity constraint			
	1	2	3	4	$p:q$	O_{max}	O_{num}	u_m
1 car radio	1	0	0	1	1:2 = 0.50	6	5	0.83
2 furry dice	0	1	0	1	2:3 = 0.66	8	6	0.75
3 power steering	1	0	1	0	1:3 = 0.33	4	6	1.50
production requirement ($pr[i]$):	2	3	4	3			$\mu =$	0.86

In the Table 2 CarSP there are $O_{num}(3) = 6$ cars requiring option 3, yet the maximum allowed is $O_{max}(3) = 4$, therefore the utility ratio $u_3 = 6/4 > 1$ and the CarSP is unsolvable.

Using option 3 we can demonstrate that we cannot satisfy the capacity constraint 1:3 in Table 2:

position i :	12	11	10	9	8	7	6	5	4	3	2	1	
option 3 position:			✓			✓			✓			✓	→ assembly line

In this schedule, we have positioned the four type 3 cars. We cannot position any of the two type 1 cars without violating the capacity constraint. Type 1 cars need to be positioned in such a way as to minimize the capacity constraint violation according to a penalty function. For each car requiring option m , there is a sub-sequence of $(q_m - 1)$ cars which follow it in a schedule, defined as an *interval of relevance* (Parrello, 1988). The penalty function assigns a penalty value to the car requiring option m , depending upon the number of cars in the interval of relevance requiring m which exceed the workstation capacity p_m . A set P of penalty values is defined for each option m and o cars requiring this option in the interval of relevance:

$$P[m, o] \text{ for } m = 1, 2, \dots, n; o = 1, 2, \dots, (q_m - 1).$$

Naturally, if the capacity constraint of option m is $p_m:q_m$, then $P[m, o] = 0$ whenever $o < p_m$. The penalty cost of option m for car i in a schedule S is calculated as:

$$\text{cost}(S, i, m) = P[m, (O[m, S_i] \times \sum_{j=i+1}^{i+(q_m-1) \leq N} O[m, S_j])] \quad (6)$$

where S_i is the type of the i th car in S . $O[m, S_i] = 1$ if car i requires option m ; 0 otherwise.

It follows, that if for all $m = 1, 2, \dots, n$ and $o = p_m + 1, \dots, (q_m - 1)$ $P[m, o] = 1$, then Equation 6 calculates the total number of options violated by a single car.

Further, it may be possible to improve the car spacing arrangement in a schedule, which can assist the workstation teams install the options. Within the interval of relevance, Parrello (1988) defines a smaller *proximity interval* for n options as, $I[1], I[2], \dots, I[n]$ where $I[m] < q_m$. If a car has a penalty cost for an option m greater than zero (i.e. $\text{cost}(S, i, m) > 0$) and if there is another car which requires m within a proximity interval $I[m]$, i.e.:

$$(\sum_{j=i+1}^{i+I[m] \leq N} O[m, S_j]) \geq 1$$

then a proximity factor $F[m]$ is added to the penalty cost for that car:

$$T_{cost}(S, i, m) = cost(S, i, m) + F[m]. \quad (7)$$

For example using $I[3]=1$ and $F[3]=7$, we find that by adding the proximity factor to the following example we can see how pressure is placed on cars requiring option 3 to have non-option 3 cars to come between them. Car 1 has car 2 requiring option 3 within $I[3]=1$ cars and car 1 has a $cost(S, 1, 3) = 2$, therefore $T_{cost}(S, 1, 3) = (2 + 7) = 9$:

position i :	...	4	3	2	1	
require option 3?	...	✗	✗	✓	✓	→ assembly line

Spacing the cars requiring the same options apart prevents the additional proximity factor from increasing the cars' penalty cost. With a proximity interval of 1, swapping the car in position 2 and car position 3 as in the following results in a cost $T_{cost}(S, 1, 3) = 2$ (since no car in the proximity interval requires option 3):

position i :	...	4	3	2	1	
require option 3?	...	✗	✓	✗	✓	→ assembly line

The cost of N cars in S with n options gives schedule cost:

$$S_{cost} = \sum_{i=1}^N \sum_{m=1}^n T_{cost}(S, i, m) \quad (8)$$

S_{cost} represents the sum of penalties for all cars which violate the capacity constraints and proximity intervals. A schedule can be derived from the problem in Table 1 with $S_{cost} = 0$.

In constraint satisfaction terms the production requirement is a *hard* constraint, and the capacity constraint is a *soft* constraint (which can be violated at a cost).

2.3 Theoretical Lower Bound

In order to test the quality of GAcSP results on unsolvable CarSPs, we have devised a method to calculate a *theoretical lower bound* for certain unsolvable CarSPs. The applicability of this lower

bound formula is limited to problems produced in the following way: solvable CarSPs (i.e. $S_{cost} = 0$) made unsolvable by a single over-utilized option. For simplicity, we make $P[m,o] = 1$ and $F[m] = 0$ for all m and o (see example in table 2).

We can calculate the required number of options in the schedule, $O_{num}(m)$, in excess of the maximum number allowed, $O_{max}(m)$, by the capacity constraint $p_m \cdot q_m$ as:

$$O_{exe}(m) = O_{num}(m) - O_{max}(m).$$

After all $O_{max}(m)$ options have been sequenced to satisfy $p_m \cdot q_m$, the remaining $O_{exe}(m)$ options need to be placed in the remaining spaces so as to minimize the capacity violation. If we add two extra options to an interval of relevance as follows:

position i :	12	11	10	9	8	7	6	5	4	3	2	1	
option 3 required:			✓			✓			✓	✓	✓	✓	→ assembly line
violation:										↑	↑	↑	$S_{cost} = 3$

where $p_3:q_3 = 1:3$, this will result in a penalty of 3 (due to the cars in positions 1 to 3), which is also the minimum violation cost (see Warwick 1995). However if we do not group the extra options in one interval of relevance, as in the following example:

position i :	12	11	10	9	8	7	6	5	4	3	2	1	
option 3 required:			✓	✓		✓			✓	✓		✓	→ assembly line
violation:				↑		↑				↑		↑	$S_{cost} = 4$

then a greater cost (in this case 4, due to positions 1, 3, 7 and 9) will result. These examples show that the grouping of extra options in available spaces in the minimum number of intervals of relevance reduces the number of violations. Consequently, $p_m - q_m$ spaces in each interval of relevance can be used to accommodate extra options. We can therefore calculate the minimum number of violation as follows:

$$\frac{O_{exe}(m)}{q_m - p_m} \times q_m.$$

In addition, an extra option m placed in a space at the end of S presents a special case where only one violation occurs, for example:

position i :	12	11	10	9	8	7	6	5	4	3	2	1	
option 3 required:	✓		✓			✓			✓			✓	→ assembly line
violation:			↑										$S_{cost} = 1$

Allowing for this special case, the lower bound formula becomes:

$$lowerbound = \frac{O_{exe}(m)}{q_m - p_m} \times q_m - p_m \quad (9)$$

3 Outline Of GAcSP

An outline of GAcSP is given in Figure 1. GAcSP is distinguished from the standard or simple GA (Goldberg, 1989) by the integration of elitism (De Jong, 1975), adaptive template type crossover (Syswerda, 1989), repairing and hill-climbing (HC) into a single strategy. The reproduction operator encourages exploitation of population information by the use of elitism and fitness biased selection. Exploration is achieved through the *uniform adaptive crossover* (UAX), which uses parent binary templates to control offspring creation. By utilizing matching parent template values (as opposed to single template crossover points as in Schaffer and Morishima (1987)) we hope to enable high fitness constraint links between parent values to be inherited.

After crossover, the offspring is repaired and hill-climbed. The repair function and HC act as mutation operators altering individual string elements. Although the GA is a robust technique for finding near optimal solutions in combinatorial search spaces, it generally lacks a local improvement ability. We provide this local ability by combining the GA with a simple string element exchange function (HC). HC increases the potential for every offspring after crossover generation, before the string is expected to compete with its peers. The combination of a GA and HC is synergistic, exploiting the abilities of each method. The representation and the operators of GAcSP are described in the following sections.

3.1 Representation

GA operators manipulate artificial chromosomes in the form of string-like data structures. PCSPs can best be handled by real-coded (Goldberg, 1990) data structures - where string positions represent PCSP variables and string elements are values from the corresponding domains. In this section, we formally define the CarSP as a PCSP. Then we shall propose a specialized GA representation for tackling this problem. We argue that this representation is applicable to PCSPs in general.

Definition 2 (CarSP):

A car sequencing problem (which we refer to as CarSP) is a partial constraint satisfaction problem (Z, D, C, g) where:

the variables are the positions in the sequence: $Z = \{1, 2, \dots, N\}$;

all variables have the same domain, which is the set of car types: $\forall x \in Z: D_x = \{1, 2, \dots, k\}$;

the set of constraints C comprises:

production requirements: $pr[i]$, where $i = 1$ to the number of car types, k , and

capacity constraints: $p_j:q_j$, where $j = 1$ to the number of options, n ;

the cost function g is S_{cost} in equation (8) with $P[m,o] = 1$ and $F[m] = 0$ for all $m = 1, \dots, n$,

and $o = p_m + 1, \dots, (q_m - 1)$.

The GAcSP objective function g maps each CarSP solution tuple to a numerical value, which is often called the *fitness*. The goal of GAcSP is to find optimal or near optimal solution tuples to the CarSP which minimize the fitness.

3.2 Reproduction Operator

The reproduction operator guides the GA through the search space by selective control using a sampling bias based upon the string fitness. The first stage of the operator implements a technique called elitism which copies the strings with the best fitness (i.e. lowest costs in CarSP) into a mating population, called the *matepool*. This technique guarantees that the elite members of the population will survive into the next generation. These best fitness strings are important because they will have low cost elements or groups of elements (i.e. *building blocks*) in their strings to pass onto their offspring, which will direct the search towards optimal regions of the search space. The second stage of reproduction involves a fitness-biased selection from the population.

<Figure 1 here>