Evolutionary Driver Scheduling with Relief Chains

Raymond S. K. Kwan rsk@comp.leeds.ac.uk
School of Computing, University of Leeds, Leeds, LS2 9JT, UK

Ann S. K. Kwan ann@comp.leeds.ac.uk
School of Computing, University of Leeds, Leeds, LS2 9JT, UK

Anthony Wren wren@comp.leeds.ac.uk
School of Computing, University of Leeds, Leeds, LS2 9JT, UK

Abstract

Public transport driver scheduling problems are well known to be NP-hard. Although some mathematically based methods are being used in the transport industry, there is room for improvement. A hybrid approach incorporating a genetic algorithm (GA) is presented. The role of the GA is to derive a small selection of good shifts to seed a greedy schedule construction heuristic. A group of shifts called a *relief chain* is identified and recorded. The relief chain is then inherited by the offspring and used by the GA for schedule construction. The new approach has been tested using real-life data sets, some of which represent very large problem instances. The results are generally better than those compiled by experienced schedulers and are comparable to solutions found by integer linear programming (ILP). In some cases, solutions were obtained when the ILP failed within practical computational limits.

Keywords

Hybrid genetic algorithms, learning property, combinatorial traits, driver scheduling, set covering, public transport.

1 Introduction

Driver scheduling for public transport, e.g., train, bus, and tram, is a significant problem because driver wages are a large cost element of public transport operations. For example, bus driver wages account for about 45% of the total operating cost (Meilton, 2001).

A *shift* is defined as a day's work for a driver. Driver scheduling is the process of compiling a set of shifts called the *driver schedule* that covers all the vehicle work. The shifts are only notional because they are not yet assigned to actual personnel. The packaging of work for actual drivers is usually performed on a weekly basis, allowing for rest days and accounting for issues such as fairness and safety regulations.

The work of one vehicle is illustrated by a *vehicle graph* as shown in Figure 1. Drivers can only be relieved at some designated places called *relief points*, which are represented by letter codes (G, S, and H). The times when vehicles are at the relief points, marked on the horizontal timescale, are known as *relief opportunities*. Between any successive pair of relief opportunities is an uninterrupted *piece of work* for the driver. Not all relief opportunities will be used for relieving drivers, and therefore, a driver may cover several contiguous pieces of work (called a *spell*) on a vehicle before being relieved.

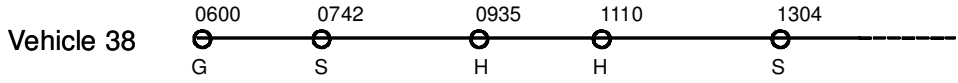

Figure 1: Portion of a vehicle graph.

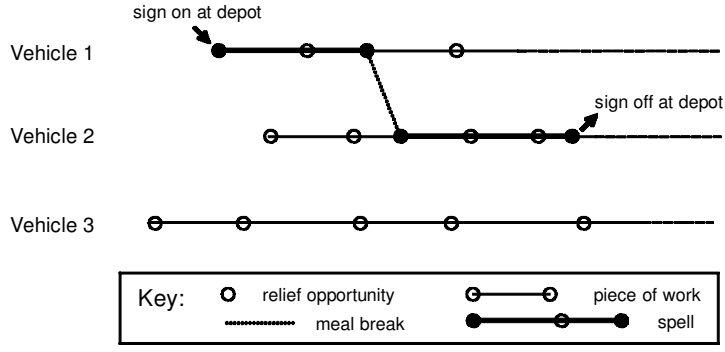

Figure 2: Illustration of a two-spell shift.

A shift typically starts with the driver signing on at a depot. The driver then works on one or more spells with breaks in between. Normally, there is at least one break long enough for taking a meal. Finally the driver returns to the depot to sign off. Figure 2 illustrates the composition of a shift in relation to some vehicle work.

The main constraints and objectives of driver scheduling are:

- every piece of work is assigned to a shift;
- the shifts must comply with all the operational constraints and labor rules;
- the total number of shifts is minimized;
- the total cost is minimized.

A good driver schedule is expected to contain some patterns of work piece and relief combinations that are keys to making the shifts involved fit seamlessly. We call such patterns and characteristics *combinatorial traits*. Combinatorial traits representing strong domain knowledge are problem dependent and very difficult to generalize, but as shown later in this paper, they might be helpful when used to guide the scheduling process.

Driver scheduling has attracted research interest since the 1960's. Wren and Rousseau (1995) gave an overview of the approaches, mainly for solving bus driver scheduling problems. Research on train driver scheduling problems has been more active since the 1990's (Kwan, 1999; Kwan et al., 1999b; Caprara et al., 1997). Many driver scheduling methods and systems have been reported in a series of international workshop conferences (e.g., Wilson (1999)).

Driver scheduling is NP-hard (Chvátal, 1979). Possible legal shifts, usually a very large set, are first generated by specific heuristics. Then, a least cost subset covering all the work is selected to form a solution schedule. Fores et al. (1999) use a blend of

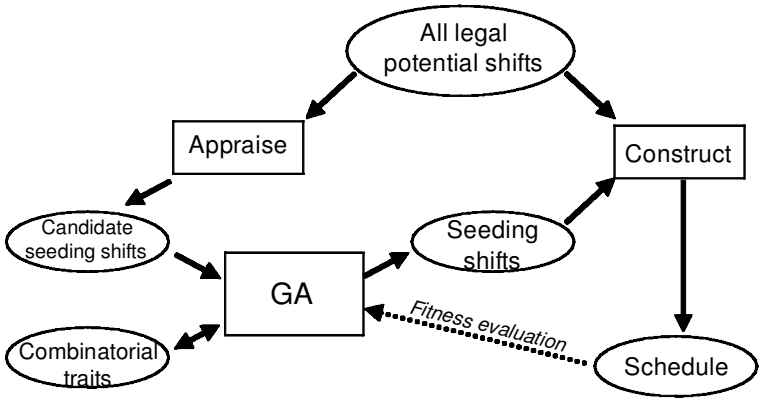

Figure 3: The GACT approach.

heuristics and Integer Linear Programming (ILP), the success and limitations of which have been discussed in Kwan et al. (2000).

In recent years, meta-heuristics (Glover and Laguna, 1993; Rayward-Smith et al., 1996) have been widely used for seeking practical near-optimal solutions to NP-hard problems. The advantages of meta-heuristics are that they are efficient in searching through very large solution spaces, always returning a solution. Also, each class of meta-heuristics has a methodical and strategic structure that is problem domain independent. GAs form a major class of meta-heuristics, which have been used for driver scheduling and related problems.

Wren and Wren (1995) worked on applying GAs for driver scheduling based on relatively small problems. Clement and Wren (1995) furthered this investigation. Both used a chromosome to represent the work pieces, the gene values indicating the selected shifts. Since several positions on the chromosome must share the same value in order to maintain integrity of individual shifts, a simple crossover operator was not feasible and specialized crossover operators were investigated. The GAs were very quick, but the resulting schedules were unsatisfactory.

A GA based on a divide and conquer approach was proposed by Gonzalez Hernandez and Corne (1996). Each problem was divided into several sub-problems by fragmenting a chromosome into smaller contiguous segments. Each sub-problem was allocated a sub-population that would evolve independently. Individuals from each sub-population were chosen and combined to form an overall solution. The chromosome representation was similar to that of Wren and Wren (1995). The divide and conquer GA showed some advantages, but it may be computationally expensive for real-world problems.

Kwan and Wren (1996) described a GA used as a pre-processor to ban many of the relief opportunities without affecting the quality of the prospective schedules, so that other driver scheduling methods could be run much faster on the reduced problem. Beasley and Chu (1996) reported some straightforward applications of GAs on the classical Set Covering problem, which is often the base model for driver scheduling methods.

A hybrid GA was proposed and preliminarily outlined in Kwan et al. (1999a). In this paper, we articulate that approach from a fresh perspective and report on further research developments and results. Our approach exploits combinatorial traits and is

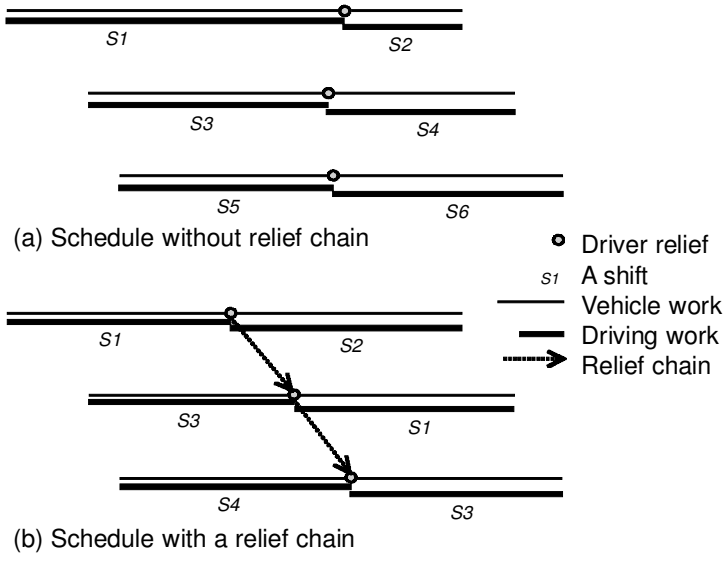

Figure 4: Relief chain example.

called the *genetic algorithm with combinatorial traits* (GACT). Figure 3 shows the overall structure of GACT highlighting the hybridization of the GA with other heuristics.

The Set Covering model is applied in this new approach. Here we use another ILP-based system (Fores et al., 1999) to provide the set of potential shifts. The main purpose of this GA is to select a small set of potential shifts as *seeding shifts* for the schedule construction process. The GA interfaces with a greedy heuristic designed to construct a complete schedule based on a given set of seeding shifts. The constructed schedule is fed back to the GA for fitness evaluation. At the same time, the group of shifts constituting a combinatorial trait called a *relief chain* (discussed in the next section) that fits efficiently in the schedule is identified and recorded as a learning property of the corresponding member in the population. That learning property is inherited by the offspring for slightly enlarging and strengthening the set of seeding shifts before the GA interfaces with the schedule construction heuristic.

2 Relief Chains: A Combinatorial Trait

An important feature in driver scheduling is that driver shifts have one or more breaks (e.g., for having a meal). After a break, the driver is usually assigned to work on another vehicle, often taking over from another driver who is due for a break. Thus a chain of events of drivers relieving each other is formed, and we call the relief opportunities involved a *relief chain*.

Well-formed relief chains, especially critical during peak periods, can avoid using extra drivers unnecessarily. This is because it is usually more efficient to relieve drivers who are due for a break with drivers who have already finished their mealbreaks. This is illustrated in Figure 4. The schedule contains work on three vehicles. In the absence of any relief chain, six drivers are needed to cover the work, whereas the schedule with a relief chain requires only four drivers.

Empirical studies (Layfield et al., 1999) have shown that good schedules are usu-

ally associated with long relief chains. Many schedules will be generated in each GA generation providing the opportunity to extract long relief chains. The shifts forming a relief chain are called *relief chain shifts*. Relief chain shifts are selected in a group rather than individually. The aim is to utilize the relief chain trait to help the next generation to construct efficient schedules.

3 Schedule Construction Heuristic

This is a greedy heuristic for constructing a driver schedule given that the GA has already provided a partial schedule consisting of some seeding shifts.

The schedule to be constructed must be valid in the sense that each piece of work is covered by at least one of the shifts chosen to form the schedule. *Overcover*, i.e., a piece of work covered by more than one shift, is allowed by the heuristic and usually can be resolved easily by manual editing before the schedule is implemented.

The heuristic can be regarded as a process of assigning a shift to cover each piece of work. First, the seeding shifts are assigned to the corresponding pieces of work covered. Each of the remaining unassigned pieces of work will have a list, called a *coverage list*, of potential shifts that can cover it. Potential shifts are assigned to the remaining uncovered pieces of work sequentially. The criterion for the next uncovered piece of work for assignment is that it has the shortest coverage list. From the coverage list, the shift that covers most other work is assigned. In the case of a tie, the shift with a lower cost is chosen.

At the beginning, many pieces of work may already be over-covered by the seeding shifts. The other shifts added during schedule construction are each chosen to cover an uncovered piece of work and therefore will not be redundant, but they may increase the amount of over-cover and cause some of the seeding shifts to become redundant. The redundant shifts are removed in two passes. First, the process will remove any redundant shifts that are not seeding shifts. In the second pass, the process will remove any redundant shifts.

4 The GACT Approach

Typically over 25,000 legal potential shifts would be input to GACT, and the final schedule may contain less than 100 shifts. The solution search space is obviously enormous, causing difficulty in finding near optimal solutions. An appraisal process therefore eliminates potential shifts that are unlikely to be useful as seeding shifts. The purpose of the GA component is to make a selection of seeding shifts. These seeding shifts are then used as the basis for forming a complete schedule by the greedy schedule construction heuristic described in Section 3. The constructed schedule is fed back to the GA for fitness evaluation of the corresponding population member and for the extraction of combinatorial traits.

A combinatorial trait critically affects schedule quality and is manifested by a group of shifts, called *CT-shifts*, that fits together. Figure 5 shows the design of the genotype and phenotype of an individual, and it illustrates the role of combinatorial traits in GACT.

Combinatorial traits act as private learning properties at the genotype level. When an offspring is produced, its chromosome and gene values represent a set of shifts selected from the pool of candidate seeding shifts. The offspring inherits combinatorial traits, and hence CT-shifts, from one of its parents. The combinatorial traits are "learned," resulting in a slightly enlarged and strengthened set of seeding shifts. The schedule construction heuristic is applied, and new combinatorial traits are extracted

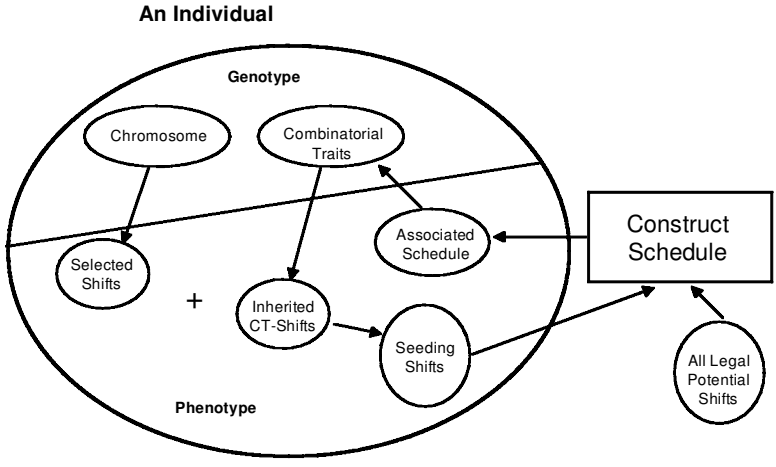

Figure 5: Role of combinatorial traits in GACT.

from the resulting schedule.

Currently GACT only exploits relief chains as a combinatorial trait. Only the longest relief chain is extracted from the newly constructed schedule. The originally inherited relief chain is no longer related to the schedule associated with the offspring and is therefore discarded. Each offspring will have its own fresh learning property (relief chain) to be passed on to its offspring. For future research, the genotype and genetic operations might be adapted to accommodate multiple relief chains evolved over many generations, such that the wider learning property base might lead to better results. But GACT adopts a simpler approach of inheriting combinatorial traits for only one generation. The evolutionary framework has the advantage that the process of identifying combinatorial traits in a schedule does not have to be perfect. If a weak combinatorial trait were extracted from a schedule associated with a fit population member, it would likely render the inheriting offspring inferior.

Section 4.1 describes a method based on the continuous solution of a relaxed LP for appraising the potential shifts to form a much smaller set of candidate seeding shifts. The main GA component of GACT is presented in Section 4.2.

4.1 Candidate Seeding Shifts from Relaxed LP Solution

The driver scheduling problem can be formulated as a Set Covering ILP. The basic model is shown below.

n	Number of legal potential shifts
m	Number of driving work pieces to be covered
c_j	Cost of shift j
a_{ij}	0-1 integer constants; 1 indicates shift j covers work piece i, and 0 otherwise
x_j	0-1 integer variables; 1 indicates shift j is used in the solution, and 0 that it is not

Minimize

$$\sum_{j=1}^{n} c_j x_j \tag{1}$$

Subject to:

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1, \quad for\ i = 1, 2, ..., m \tag{2}$$

$$x_j = 0\ or\ 1, \quad for\ j = 1, 2, ..., n \tag{3}$$

Objective (1) minimizes the total cost. Constraint (2) ensures that each piece of work is covered by at least one driver. Constraint (3) requires that whole shifts are used. In practice, the model is extended to cater to other practical objectives and constraints (Fores et al., 1999).

Integer solution techniques, e.g., branch-and-bound, are limited by the amount of search space that can be explored within reasonable time. For this reason, ILP solution processes may sometimes have to be terminated before any integer solution is found. However the relaxed LP, i.e., ignoring the integer constraints, can usually be solved quickly. The non-integer relaxed LP solution has been shown to contain an average of 79% of the shifts forming the integer solution yielded by the full ILP (Kwan et al., 1999a).

In the relaxed LP solution, the values of the shift variables lie between 0 and 1. It has been observed that the closer to a value of 1, the higher the chance the shift will be used in the final solution schedule (Kwan et al., 1999a). Hence all the shifts in the relaxed LP solution that have their solution values greater than a given parameter (e.g., 0.3) are included in the candidate seeding shift set. Note that the schedule construction algorithm selects from the large original input set of potential shifts. To improve computational speed, the large input set of potential shifts is reduced by banning those shifts that do not use the relief opportunities used by the candidate seeding shifts.

Others have utilized the relaxed LP solution in solving scheduling problems by means of Constraint Programming (Guerinik and Caneghem, 1995; Rodosek et al., 1996; Curtis et al., 1999). They use the fractional values of the relaxed LP solution as a guide for choosing the variables to be satisfied. Curtis et al. (1999) differed from Guerinik and Rodosek (1995) in that the fractional values are first assigned to relief opportunities, whereas the latter use the fractional values of individual shifts in the relaxed LP solution.

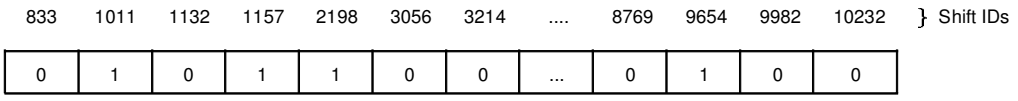

Figure 6: Chromosome representation.

4.2 The GA Component

The main design of the GA component in GACT is presented in this section. The GA uses a fixed size population with replacement between generations. It also employs a special adaptive mutation scheme, which will be discussed in Section 4.2.1.

Chromosome Representation

Figure 5 has already outlined the genotype and phenotype design. In the genotype, there is a single chromosome, which is represented by a 0-1 binary string. Each gene determines whether a particular candidate seeding shift is selected (allele value = 1) or not (allele value = 0). Figure 6 shows an example in which the appraisal process has eliminated all the given potential shifts except shifts 833, 1011, 1132, 1157, etc. These remaining shifts are the candidate seeding shifts. The example chromosome has selected shifts 1011, 1157, etc. as actual seeding shifts. The number of genes to be assigned a value 1 is restricted to no more than the sum of the shift variables in the relaxed LP solution, which has been shown to be a very accurate estimate of the minimum number of shifts in the integer solution.

Fitness Function

The fitness of a population member in GACT is assessed based on the total cost C of its associated schedule:

$$C = \sum_{j=1}^{n} (c_j + K)$$

where n = number of shifts in the schedule, c_j is the cost of shift j, and K is a large constant. In most driver scheduling problems, the most important objective is to minimize the number of shifts, hence a large constant K per shift is added to give this objective priority. A similar objective function is used by an ILP-based method (results are compared in Section 5). Therefore K has been set to 5000, which is the same used in the ILP. Obviously, the lower the schedule cost, the fitter the population member.

Crossover with Inheritance and Population Replacement

The standard roulette wheel method is used for selecting parents for crossover. Offspring are not chosen for reproduction in the current generation. The number of crossovers per generation has been arbitrarily set at half the population size. After a generation has completed, the parent population and the child population are merged and ranked by fitness. Only a parameterized number of the fittest individuals from the merged population are allowed to survive into the next generation. A fixed population size is maintained at the beginning of a generation by filling up any shortfall in the population with randomly generated individuals.

Three types of crossover operators have been investigated:, single-point crossover, two-point crossover, and uniform crossover. Table 1 compares the results obtained for

Table 1: Comparative results for different crossover operators.

Data	No. of shifts (cost in hh:mm paid time)		
	Single Point	Two Point	Uniform
T3	74* (561:01)	74 (562:23)	74 (562:10)
B8	48* (525:13)	48 (531:37)	49 (537:19)
T1	14* (124:23)	15 (128:45)	15 (128:54)
B1	34* (294:05)	34* (294:05)	34* (294:05)
T2	42* (301:52)	42 (301:56)	42 (302:24)
B2	35* (295:18)	35 (299:58)	35 (298:38)
B3	29* (266:29)	30 (269:05)	30 (271:40)
R1	50* (419:06)	51 (424:15)	51 (426:44)
T7	116* (1005:57)	116 (1013:47)	116 (1014:32)
T4	62* (507:39)	63 (514:06)	63 (517:58)
B5	48* (508:06)	49 (513:04)	48 (519:05)

* Solution with least number of shifts and lowest cost

eleven sets of data.

We adopt single-point crossover because it consistently yields the best results among the three crossover operators. Uniform crossover appears to be marginally worse than the other two. We attempt to offer an explanation as follows, although it may not be possible to give any concrete proofs. Akin to the notion of combinatorial traits, there are many significantly well-fitted groups among the potential shifts. Since each potential shift has multi-faceted interactions with other shifts in covering the work, it may belong to many well-fitted groups. With a schedule construction heuristic that builds upon a partial schedule, its success would depend on the presence within the partial schedule of at least some rudiments of some well-fitted groups. In GACT, the elite chromosomes represent such rudiments for seeding the schedule construction process. Any crossover operation would have the effect of breaking up such rudiments to some extent. Uniform crossover seems to have the greatest impact of fragmenting the bit strings of the parent chromosomes, and therefore the rudiments of well-fitted groups may be broken up to a larger extent and cause marginally poorer results.

After crossover, one child inherits the relief chain shifts from one of the parents chosen randomly, and the other child inherits the relief chain shifts from the other parent. For each child, its inherited relief chain shifts, merged with the candidate seeding shifts selected by its genes, are input to the greedy schedule construction heuristic. The constructed schedule becomes part of the child's phenotype, and the longest relief chain in that schedule becomes the relief chain trait of the child.

4.2.1 Adaptive Mutation

Mutation may take place after a crossover. The mutation rate is variable, mainly dependent on the convergence rate of the GA. In early generations, the GA is less likely to be trapped at local optima, and the rate of mutation should therefore be kept low so that the population will evolve progressively. The rate of mutation should then gradually increase to a high level when convergence is near. Since GACT only mutates the children and there is an elitist replacement strategy, the pressure of selecting elite parents

Table 2: Effect of adaptive mutation (M = 5).

Generations completed	10	20	30	40	50	60	70	80	90	100	150
Max. Generations (g)	**Accumulated number of mutations**										
50	3	14	20	28	44	-	-	-	-	-	-
100	2	6	13	18	30	31	32	45	75	104	-
150	2	3	5	12	22	23	26	31	35	36	72

Table 3: Number of mutations for different M values.

Generations completed	Accumulated number of mutations			
	M=1	M=5	M=10	M=20
10	12	3	2	1
20	56	14	6	2
30	85	20	13	3
40	139	28	18	8
50	243	44	30	8

is maintained throughout the evolution.

The mutation scheme used in GACT is aggressive because the criterion to mutate an offspring is when the gene has the same value for both parents rather than in the entire population. The objective is to bring into the population new individuals dissimilar to the parents by favoring the child to have an allele value at each locus different from at least one of its parents. This has the effect of broadening the solution search. Since aggressive mutation brings in new individuals to the population continuously, a small population size is sufficient to give good results, and consequently, the computational performance of the GA is improved.

The mutation rate depends on three parameters: a control value M, the number of generations the GA process will run g, and the number of generations the fittest member has survived s. After each successful mating, a random number r is generated with value between 1 and $(g * M)$. If $r \leq s$, the offspring will be subject to mutation.

The mutation rate is probabilistic, and the chance to mutate increases if s increases. A high s value indicates that the fittest member has become dominant in the GA process, and mutation should therefore be intensified to introduce new species. The chance of mutation is less if either or both M and g are large.

Table 2 shows for a sample problem instance the mutation rate at different numbers of generations with $M = 5$ and different maximum numbers of generations the GA is set to run. Table 3 shows the effect of using different M values on a particular problem for a total of 50 generations.

When mutation is enabled for an offspring, those genes for which its parents both have the same value will be noted. Some of these genes belonging to the offspring will be selected at random for mutation. A gene is mutated by switching a 0 to a 1 and vice versa.

Table 4: Comparative test results.

Data	Work pieces	Shifts variables	Solution schedule size (no. of shifts)			Cost comparison **	Elapsed time GACT	Elapsed time ILP (B&B only)	Relief chain size ***	
			Manual	ILP	GACT	%	(seconds)	(seconds)	ILP	GACT
B1	127	3560	34	34	34	2.02	36	22	6	7
B2	154	11817	34	34	35	2.00	22	84	7	6
T1	158	16379	-	14	14	0.40	14	185	6	6
T2	196	12343	-	42	42	0.13	0.4	13	5	4
T3	236	11950	76	74	74	-0.08	41	4	2	2
B3	254	74816	31	29	29	2.28	78	10	8	11
B5	338	31394	49	47	48	4.40	65	36	10	7
T4	340	25000	68	Fail	62	-	955	Large	-	7
B8	345	29999	49	Fail	48	-	231	Large	-	12
T5	359	29380	-	89	90	2.2	303	724	10	8
T6	400	16636	-	50	51	1.23	97	34	9	9
T7	434	25099	116	116	116	0.20	80	69	5	4
B4	461	22568	-	Fail	74	-	452	Large	-	6
R1	483	29500	50*	49	50	2.52	96	139	7	7
R2	483	6437	50*	49	51	4.11	14	24	7	6
B7	485	112546	73*	73	73	2.2	290	118	9	12
T8	537	14818	115	112	113	0.71	263	20	7	8
B6	557	73068	60	58	60	3.35	362	22	9	9
T9	1108	29465	-	276 +	269	-	9352	N/a	-	8
T10	1438	28639	-	349 +	341	-	13333	N/a	-	7

\+ Combined result of subdivided problems
* Manual schedule has violated the labor agreement rules
** Percentage increase in schedule cost of GACT against ILP
*** No. of relief chain shifts in the longest relief chain in the solution

Experiments have been carried out to test the effect of the opposite strategy of using a high mutation rate early on and then lowering the mutation rate at later generations. It was found that the alternative process converges quicker, but its solutions are worse. The eleven data sets in Table 1 were used for the experiments, four of which converge to solutions with a higher number of shifts. The rest yielded solutions as good in terms of number of shifts, but were worse in cost.

5 Results

GACT has been implemented and tested on a Pentium II 333 MHz PC with 196 megabyte RAM. Comparative results on twenty real-life problems from the public transport industry are presented in Table 4. Problem instances prefixed by B are bus problems, T are train problems, and R are tram problems. The GACT results are compared with those compiled manually by the transport operators and those obtained by a specialized Set Covering ILP.

The two larger problems T9 and T10 have been tested using population sizes of 100, 200, 300, 400, and 500. For the other problems, population sizes of 25, 50, 100, 200, and 300 were tried. Smaller population sizes would reduce the computation for each generation, but they might not result in very good solutions. Hence some larger population sizes were also tried. On average, each test has been run ten times using different combinations of parameters. The most effective population sizes were found

Table 5: Summary of results of ten runs with fixed parameters.

Data	Work pieces	No. of shifts*					Cost			
		In Table 4	-1	=	+1	+2 or more	Ave.	Min.	Max.	Std. Dev.
B1	127	34		5	5		295	292	297	1.42
B2	154	35		8	2		297	293	303	3.20
T1	158	14		2	8		127	125	130	1.28
T2	196	42		10			302	302	302	0.27
T3	236	74		10			561	561	562	0.61
B3	254	29		9	1		269	266	273	2.16
B5	338	48		5	5		516	507	521	5.22
T4	340	62		6	4		509	505	514	3.14
B8	345	48		5	5		528	512	541	10.05
T5	359	90		6	4		796	791	812	7.68
T6	400	51	1	8	1		411	408	417	2.64
T7	434	116		4	5	1	1022	1006	1036	11.13
B4	461	74		6	3	1	841	816	851	9.87
R1	483	50		6	4		423	417	426	2.97
R2	483	51	2	8			437	431	442	3.48
B7	485	73		3	7		894	886	903	5.61
T8	537	113		10			889	883	894	3.70
B6	557	60		4	6		680	670	695	8.40
T9	1108	269		2		8	2119	2106	2132	7.68
T10	1438	341	1	4	3	2	2704	2687	2736	17.50

* The four columns on the right show the distribution of runs comparing the number of shifts with the solutions found before these runs as shown in Table 4.

to be 400 to 500 for T9 and T10, and 100 to 300 for the other problems.

Fixing the parameters used above, each data set was run ten times by varying the pseudo random number seed at the beginning of each run. The results are summarized in Table 5.

Table 5 shows that GACT is quite robust. Comparing the number of shifts in the ten runs with the best solution found before the runs, on average 62.5% of the runs have the same or better results. In terms of solution costs, the variations between the runs are small. Apart from the largest problem that has the largest standard deviation in cost, there is no obvious trend detected.

Kwan (1999) has found from experiments employing exhaustive search that the distribution of optimal and near optimal solutions is acutely skewed for the driver scheduling problem. In one particular experiment with about 25,000 potential shifts, there were only five optimal solutions using 14 shifts. There would be an enormous number of ways to replace one or more of the shifts in the 14-shift solution to give rise to solutions with 15 or more shifts. It is suspected that for that particular data set, there are thousands of 15-shift solutions and millions of 16-shift solutions. Since it is very

difficult to find the optimal or very near optimal solutions, we are more interested in the best result than the average result in the test runs.

The majority of the problem instances are complex. In some cases, the ILP process being compared fails to find an integer solution after a large number of nodes of the branch-and-bound search tree has been explored. In these circumstances, the target is raised by one shift and the ILP is re-run. The process is repeated if an integer solution still cannot be found. In three instances, T4, B4, and B8, the process was abandoned after the target had been raised many times without success. T9 and T10 are very large problem instances, which had to be decomposed into smaller sub-problems and solved independently by the ILP process. The ILP results shown are for the combined results of individual sub-problems. No manual solutions are available for these large instances.

Both GACT and the branch-and-bound phase of the ILP process take the relaxed LP solution as a starting point, and therefore the elapsed times for them are compared. In instances T7, T9, and T10, the ILP has been run repeatedly with increasing targets due to difficulty in finding an integer solution. The time for each failed run varies from less than half an hour for T7 to about two hours for T10.

The results have shown that GACT can achieve very good solutions. In many cases, it achieves solutions that are as good as the ILP process in terms of number of shifts. The costs of the GACT solutions are, in general, higher than those of the ILP solutions. This may be because the fitness function used in GACT is biased toward minimizing the number of shifts. When the ILP process is used, very large problems may have to be decomposed into smaller sub-problems and solved independently. In contrast, GACT can be used to solve each large problem in one go. Also, GACT has the advantage of always being able to produce a solution, whereas the ILP process in some cases may need several runs with increasing target numbers of shifts and may still not be able to find any integer solution.

In most cases where a manual solution is known, the ILP or GACT finds a solution at least as good, and often better. All except one (B2) of the solutions (in Tables 4 and 5) obtained by GACT have smaller or equal number of shifts compared with the known manual solutions. The manual schedule for B2 contains shifts that violate the given labor agreement rules.

The performance of GACT is very similar to the branch-and-bound phase of the ILP process. The largest problem takes just under 4 hours, and the rest take less than an hour to run. The comparison has not included the time taken by the failed ILP runs before the final results were obtained.

The size (in terms of number of relief chain shifts) of the longest relief chain in the GACT solution is relatively small compared to the number of shifts in the solution. Hence although the relief chain shifts may be critical, they are not dominating in number during the schedule construction process. There is similarity in the lengths of the longest relief chains found in the GACT solution and in the corresponding ILP solution, which indicates a correlation between the relief chain trait and the good performance of GACT. In a separate set of experiments, the relief chain trait feature was disabled in GACT. The solution schedules were worse by one shift for nine of the data sets (T1, B8, T6, T7, T4, T9, T10, B5, B6), and worse with higher costs in seventeen. Also, the run times were slightly higher.

6 Conclusions

The GACT approach has set out a framework whereby the power of search heuristics is augmented by domain knowledge in a structured manner. Combinatorial traits represent strong domain knowledge, but they are difficult to generalize. The scheme of inheritance and application of combinatorial traits has lessened the burden of accurately identifying and assessing the traits, because any bogus traits would weaken and reduce the chance of survival of the offspring.

The main advantages of GACT are that it can always produce feasible good quality solutions even in cases that are difficult for ILP to solve, and that it can solve very large problem instances without the need to subdivide them. In many instances, the results are better than the manually compiled solutions. Many of the GACT results are at least as good as those obtained by ILP.

The GACT approach has some other general advantages. Being heuristic, the Construct process (in Figure 3) is flexible for adaptation to handle extra scheduling constraints. Problem specific rules may be inserted into the heuristic easily, but it would generally be more difficult for ILPs. At present, the Appraise process is implemented based on the relaxed LP solutions. It is possible that a faster heuristic could be developed instead. The modular structure of the approach makes it easy to test and replace the component modules.

Exploitation of combinatorial traits is currently limited to relief chains; additional combinatorial traits may be incorporated into GACT in the future. For example, Kwan et al. (1999a) suggested overlapping work pieces as a possible combinatorial trait for further investigation. When the GA converges prematurely, more shifts than the minimum are used and often some pieces of work are persistently covered by multiple shifts. By analyzing the shifts covering such overlapping work pieces, some suspected bad linkages might be identified and banned in the schedule construction process. Furthermore, inheritance of the relief chain combinatorial trait is only limited to one generation. Further research may involve a more sophisticated design of the genotype and genetic operations so that several or more relief chains can be preserved, evolved, and utilized over many generations.

Another current limitation of GACT is that often many trial runs with different parameters and random number seeds are needed. Further research will fine-tune the parameters for GACT and help to develop a practical strategy for deploying multiple runs to ensure a good chance of obtaining the best results. Also, it might be useful to apply a local search to improve the solutions obtained by GACT.

Acknowledgments

The authors would like to thank the Engineering and Physical Sciences Research Council for funding this research and all the transport operators who have provided many challenging problems over the years.

References

Beasley, J. E. and Chu, P. C. (1996). A genetic algorithm for the set covering problem. *European Journal of Operational Research*, 94:392–404.

Caprara, A. et al. (1997). Algorithms for railway crew management. *Mathematical Programming*, 79:25–141.

Chvátal, V. (1979). A Greedy Heuristic for the Set-Covering Problem. *Mathematics of Operations Research*, 4:233–235.

Clement, R. and Wren, A. (1995). Greedy genetic algorithms, optimising mutations and bus driver scheduling. In Daduna, J. R., Branco, I., and Paixao, J. M. P., editors, *Computer-Aided Transit Scheduling*, pages 213–235. Springer Verlag, Berlin, Germany.

Curtis, S. D., Smith, B. M., and Wren, A. (1999). Forming bus driver schedules using constraint programming. In *Proceedings of First International Conference on the Practical Applications of Constraint Technologies and Logic Programming*, pages 239–254, The Practical Application Company, Blackpool, UK.

Fores, S., Proll, L. G., and Wren, A. (1999). An improved ILP system for driver scheduling. In Wilson, N. H. M., editor, *Computer-Aided Transit Scheduling*, pages 43–61. Springer-Verlag, Berlin, Germany.

Glover, F. and Laguna, M. (1993). Tabu search. In Reeves, C. R., editor, *Modern heuristic techniques for combinatorial problems*, pages 70–150. Blackwell Scientific, Oxford, UK.

Gonzalez Hernandez, L. F. and Corne, D. W. (1996). Evolutionary divide and conquer of the set covering problem. In Fogarty, T. C., editor, *Evolutionary Computing*, pages 198–208. Springer, Berlin, Germany.

Guerinik, N. and Caneghem, M. V. (1995). Solving crew scheduling problems by constraint programming. In Montanari, U. and Rossi, F., editors, *Principles and Practice of Constraint Programming CP'95*, pages 481–498. Springer, Berlin, Germany.

Kwan, A. S. K. (1999). *Train driver scheduling*. Doctoral Thesis, University of Leeds, Leeds, UK.

Kwan, R. S. K., and Wren, A. (1996). Hybrid genetic algorithms for bus driver scheduling. In Bianco, I., and Toth, P., editors, *Advanced Methods in Transportation Analysis*, pages 609–619, Springer, Berlin, Germany.

Kwan, A. S. K., Kwan, R. S. K., and Wren, A. (1999a). Driver scheduling using genetic algorithms with embedded combinatorial traits. In Wilson, N. H. M., editor, *Computer-Aided Transit Scheduling*, pages 81–102. Springer-Verlag, Berlin, Germany.

Kwan, A. S. K. et al. (1999b). Producing train driver schedules under different operating strategies. In Wilson, N. H. M., editor, *Computer-Aided Transit Scheduling*, pages 129–154. Springer-Verlag, Berlin, Germany.

Kwan, R. S. K., Kwan, A. S. K., and Wren, A. (2000). Hybrid genetic algorithms for scheduling bus and train drivers. In *Proceedings of IEEE Congress on Evolutionary Computation*, pages 285–292, IEEE Press, Piscataway, New Jersey.

Layfield, C. J., Smith, B. M., and Wren, A. (1999). Bus relief opportunity selection using constraint programming. In *Proceedings of First International Conference on Practical Application of Constraint Technologies and Logic Programming*, pages 537–552, The Practical Application Company, Blackpool, UK.

Meilton, M. (2001). Selecting and implementing a computer aided scheduling system for a large bus company. In Voß, S. and Daduna, J. R., editors, *Computer-Aided Scheduling of Public Transport*, pages 203–214. Springer, Berlin, Germany.

Rayward-Smith, V. J. et al., editors. (1996). *Modern heuristic search methods.* Wiley, Chichester, UK.

Rodosek, R., Wallace, M. G., and Hajian, T. (1996). A new approach to integrate mixed integer programming. In Freuder, E. C., editor, *CP'96 workshop on Constraint Programming Applications: An Inventory and Taxonomy*, Springer, Berlin, Germany.

Wilson, N. H. M., editor (1999). Computer-Aided Transit Scheduling. In *Proceedings of the Seventh International Workshop on Computer-Aided Scheduling of Public Transport*, Springer, Berlin, Germany.

Wren, A. and Rousseau, J.-M. (1995). Bus driver scheduling: An overview. In Daduna, J. R., Branco, I., and Paixao, J. M. P., editors, *Computer-Aided Transit Scheduling*, pages 173–187. Springer, Berlin, Germany.

Wren, A. and Wren, D. O. (1995). A genetic algorithm for public transport driver scheduling. *Computers and Operations Research*, 22:101–110.