Pergamon

PII: S0305-0548(96)00032-9

Computers Ops Res. Vol. 24, No. 1, pp. 17–23, 1997

0305-0548/96 $17.00+0.00

A GENETIC ALGORITHM FOR THE GENERALISED ASSIGNMENT PROBLEM

P. C. Chu†‡ and J. E. Beasley§
The Management School, Imperial College, London SW7 2AZ, England

(Received January 1996; in revised form April 1996)

Scope and Purpose—The generalised assignment problem is an important problem, that, judging by the number of articles concerned with it, has attracted a fair amount of interest from Operations Research workers. Both exact (optimal) and heuristic approaches have been presented in the literature. This article shows that genetic algorithms, an approach that arose with computer science, can be applied successfully to a problem that has traditionally been associated with Operations Research. The article demonstrates clearly that genetic algorithms are capable of producing better quality results for the generalised assignment problem than traditional Operations Research heuristic approaches (albeit at greater computational cost). This result should be of interest, not only to those interested in the generalised assignment problem, but also to others involved in developing heuristics for integer programming problems. In particular, this article illustrates that, applied correctly, genetic algorithms can provide the best possible heuristic solution approach for such problems. In order to advance the state of the art in algorithms for the generalised assignment problem a benchmark set of relatively large test problems is solved and made publically available to others.

Abstract—In this paper we present a genetic algorithm (GA)-based heuristic for solving the generalised assignment problem. The generalised assignment problem is the problem of finding the minimum cost assignment of n jobs to m agents such that each job is assigned to exactly one agent, subject to an agent's capacity. In addition to the standard GA procedures, our GA heuristic incorporates a problem-specific coding of a solution structure, a fitness-unfitness pair evaluation function and a local improvement procedure. The performance of our algorithm is evaluated on 84 standard test problems of various sizes ranging from 75 to 4000 decision variables. Computational results show that the genetic algorithm heuristic is able to find optimal and near optimal solutions that are on average less than 0.01% from optimality. The performance of our heuristic also compares favourably to all other existing heuristic algorithms in terms of solution quality. Copyright © 1996 Elsevier Science Ltd

1. INTRODUCTION

The generalised assignment problem (GAP) is a well-known, NP-complete combinatorial optimisation problem which involves finding the minimum cost assignment of n jobs to m agents such that each job is assigned to exactly one agent, subject to an agent's available capacity. Recent extensive reviews of applications of the GAP and the existing exact and heuristic algorithms can be found in Cattrysse and Van Wassenhove [1] and Osman [2] and will not be repeated here. The existing exact algorithms are only effective in certain GAP instances where the constraints are loose. For the more difficult highly-capacitated problems, exact algorithms can only solve problems involving up to a few hundred decision variables before the search trees grow prohibitively large. Thus larger-sized problems are often tackled by applying heuristics to obtain approximate solutions. This article proposes a heuristic method based on genetic algorithms.

A genetic algorithm (GA) is an 'intelligent' probabilistic search algorithm which simulates the process of evolution by taking a population of solutions and applying genetic operators in each reproduction. Each solution in the population is evaluated according to some fitness measure. Highly fit solutions in the population are given opportunities to reproduce. New 'offspring' solutions are generated and unfit solutions in the population are replaced. This evaluation-selection-reproduction cycle is repeated until a satisfactory solution is found. Examples for which GAs have been applied successfully to combinatorial optimisation problems are the set covering problem [3] and the set partitioning problem [4].

In addition to the standard GA procedures, our GA heuristic also incorporates a problem-specific

† To whom all correspondence should be addressed (email: p.chu@ic.ac.uk)
‡ Paul. C. Chu received a Bachelor's degree in Electrical Engineering from Cornell University in 1992 and a Master's degree in Operations Research and Industrial Engineering also from Cornell University in 1993. He is currently doing a PhD in the Management School (Operational Research Group) at Imperial College.
§ J. E. Beasley has a BA in Mathematics from Cambridge University and an MSc and PhD in Management Science from Imperial College, London. He is a member of the academic staff at Imperial College, London. His research interests are in linear optimisation (principally integer programming/combinatorial optimisation) and he is the author of some fifty papers.

representation scheme, separated fitness and unfitness evaluation functions and a local improvement procedure. Our empirical testing shows that the quality of the solutions generated by the GA heuristic is superior to all existing heuristic methods, albeit at greater computational expense compared with some heuristics. Larger-sized problems up to 20 agents and 200 jobs with different levels of difficulty are also generated and benchmark results for these are given using our GA heuristic.

2. THE GENERALISED ASSIGNMENT PROBLEM (GAP)

Let $I = \{1,2,...,m\}$ be a set of agents, and let $J = \{1,2,...,n\}$ be a set of jobs. For $i \in I$, $j \in J$, define c_{ij} as the cost of assigning job j to agent i (or assigning agent i to job j), r_{ij} as the resource required by agent i to perform job j, and b_i as the resource availability (capacity) of agent i. Also, x_{ij} is a 0–1 variable that is 1 if agent i performs job j and 0 otherwise. The mathematical formulation of the GAP is:

$$\text{Minimise} \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \tag{1}$$

$$\text{Subject to} \sum_{i \in I} x_{ij} = 1, \forall j \in J \tag{2}$$

$$\sum_{j \in J} r_{ij} x_{ij} \leq b_i, \forall i \in I \tag{3}$$

$$x_{ij} \in \{0,1\}, \forall i \in I, \forall j \in J \tag{4}$$

(2) ensures that each job is assigned to exactly one agent and (3) ensures that the total resource requirement of the jobs assigned to an agent does not exceed the capacity of the agent.

3. THE GA HEURISTIC

Genetic algorithms deal with a population of solutions and tend to manipulate each solution in a simple way. In a GA a potential solution to a problem is represented as a set of parameters known as a gene. These parameters are joined together to form a string of values known as a chromosome. A good representation scheme is important in a GA and it should clearly define meaningful crossover, mutation and other problem-specific operators such that minimal computational effort is involved in these procedures. To meet this requirement, we use an efficient representation in which the solution structure is an ordered structure (n-dimensional vector) of integer numbers. These integer numbers identify the agents, as assigned to vector elements denoted by the jobs (see Fig. 1). This representation ensures that all the equality constraints in (2) are automatically satisfied since exactly one agent is assigned to each job. However this representation does not guarantee that the capacity constraints in (3) will be satisfied. The steps involved in our GA heuristic for the GAP are as follows:

1. Generate an initial population of N randomly constructed solutions. Each of the initial solutions is generated by randomly assigning an agent to each job. Note that since the initial solutions may violate the capacity constraints in (3), initial solutions may be infeasible. Let s_{kj} represent the agent assigned to job j ($j=1,...,n$) in solution k ($k=1,...,N$).
2. Decode the solution structure to obtain the fitness value. The fitness of a solution is calculated according to a given fitness function. Conventionally it must take into account not just the cost of a solution, but also the degree of infeasibility of a solution. Rather than penalise the fitness (or the objective function) when a solution is infeasible, which is a common approach in GAs, we adopt the approach used in [4] and associate two values with each solution. One of these values is called fitness and the other is called unfitness. The fitness f_k of solution k is equal to its objective function value as calculated by

$$f_k = \sum_{j \in J} c_{s_{kj}j} \tag{5}$$

job	1	2	3	4	5	...	$n-1$	n
agent	2	1	3	m	3	...	1	2

Fig. 1. Representation of an individual's chromosome.

The unfitness u_k of solution k is a measure of infeasibility (in relative terms) as calculated by

$$u_k = \sum_{i \in I} \max\left[0, \left(\sum_{j \in J, s_{ij}=i} r_{ij}\right) - b_i\right] \tag{6}$$

and note here that $u_k = 0$ if and only if solution k is feasible.

3. Select two parent solutions for reproduction. We use the binary tournament selection method. In a binary tournament selection, two individuals are chosen randomly from the population. The more fit (smaller fitness value) individual is then allocated a reproductive trial. In order to produce a child, two binary tournaments are held, each of which produces one parent. Note that the selection criteria does not involve the unfitness value of an individual.

4. Generate a child solution by first applying a crossover operator to the selected parents. We use the simple one-point crossover operator, in which a crossover point $p \epsilon J$ is selected randomly and the child solution will consist of the first p genes taken from the first parent and the remaining $(n-p)$ genes taken from the second parent, or vice versa with equal probabilities. The crossover procedure is followed by a mutation procedure. This mutation procedure involves exchanging elements in *two* randomly selected genes (i.e. exchanging assigned agents between two randomly selected jobs). Computational experience (see Section 4) showed that the GA involving only the crossover and mutation operators is effective in producing good-quality solutions. But the algorithm can be further improved by using a problem-specific heuristic operator which involves two local improvement steps as described below.

(a) Let T_j represent the agent assigned to job j in the child solution. For each agent $i \epsilon I$, if the resource capacity of agent i is exceeded $\left(\sum_{j \in J, T_j = i} r_{ij} > b_i\right)$, then a single randomly selected job q with $T_q = i$ is reassigned from agent i to the next agent (in the order of $i+1,...,m, 1,...,i-1$) that has adequate remaining capacity (if one can be found).

(b) For each job $j \epsilon J$, search for $\min_{i, i \in I} c_{ij}$ that satisfies both $c_{ij} < c_{T_j j}$ and $\left(\sum_{q \in J, T_q = i} r_{iq}\right) + r_{ij} \leq b_i$. If such an i can be found, then reassign job j from agent T_j to agent i.

Steps (a) and (b) are performed only once in that order. Without further violating the constraints in (2) and (3), step (a) attempts to improve the feasibility of the child by reassigning jobs from overly-capacitated agents to less-capacitated agents and step (b) attempts to improve the cost of the child by reassigning jobs to agents with lower cost. In terms of computational complexity, step (a) takes $O(m^2 n)$ operations whilst step (b) takes $O(mn)$ operations. Hence, the heuristic operator has a complexity of $O(m^2 n)$. Note that this is a worst-case analysis which does not take into consideration any special data structures. Our computational experience shows that in actual practice, although step (a) may seem more expensive than step (b), the actual computing time in step (a) is less than that in step (b).

5. Replace an individual in the population by the child solution. In our population replacement scheme, the individual in the population with the highest *unfitness* value (i.e. the most infeasible solution) is replaced by the child. If the population consists of all feasible solutions ($u_k = 0$, $\forall k$), the individual with the highest fitness is replaced. This replacement scheme helps to eliminate infeasible solutions in the population. Note that a duplicate child, defined as a solution whose structure is identical to any of the solution structures already in the population, is not allowed to enter the population because otherwise a population may come to consist of all identical solutions, thus severely limiting the GA's ability to generate new solutions.

6. Steps 3–5 are repeated until M non-duplicate children have been generated without improving the best solution found so far.

4. COMPUTATIONAL RESULTS

The algorithm presented was coded in C and run on a Silicon Graphics Indigo (R4000, 100 MHz). The algorithm was tested on 84 test problems ranging from 5 agents/15 jobs to 20 agents/200 jobs. All of these test problems are publicly available electronically; email the message "gapinfo" to 'o.rlibrary@ic.ac.uk' for detailed information. Of the 84 problems we used, 60 problems are categorised as 'small-sized' problems. They are taken from the literature and were used in the simulated annealing heuristic of Cattrysse [5], the set partitioning heuristic of Cattrysse, Salomon and Van Wassenhove [6] and the hybrid simulated annealing/tabu search heuristic of Osman [2]. These 'small-sized' problems have the following characteristics:

Table 1. Computational results for 'small-sized' problems

Prob	Optimal	Best sol'n in each of the 10 trials										Avg. % σ	Avg. Best Sol'n Time	Avg. Exec. Time
gap1-1	336	o	o	o	o	o	o	o	o	o	o	0	0.4	74.4
gap1-2	327	o	o	o	o	o	o	o	o	o	o	0	1.0	74.7
gap1-3	339	o	o	o	o	o	o	o	o	o	o	0	0.1	69.9
gap1-4	341	o	o	o	o	o	o	o	o	o	o	0	0.2	71.9
gap1-5	326	o	o	o	o	o	o	o	o	o	o	0	0.3	71.0
gap2-1	434	o	o	o	o	o	o	o	o	o	o	0	0.2	79.2
gap2-2	436	o	o	o	o	o	o	o	o	o	o	0	1.8	86.2
gap2-3	420	o	o	o	o	o	o	o	o	o	o	0	3.2	76.9
gap2-4	419	o	o	o	o	o	o	o	o	o	o	0	2.9	77.3
gap2-5	428	o	o	o	o	o	o	o	o	o	o	0	0.3	80.2
gap3-1	580	o	o	o	o	o	o	o	o	o	o	0	5.0	84.3
gap3-2	564	o	o	o	o	o	o	o	o	o	o	0	0.5	83.7
gap3-3	573	o	o	o	o	o	o	o	o	o	o	0	1.0	86.8
gap3-4	570	o	o	o	o	o	o	o	o	o	o	0	0.3	85.0
gap3-5	564	o	o	o	o	o	o	o	o	o	o	0	4.1	85.5
gap4-1	656	o	o	o	o	o	o	o	o	o	o	0	1.7	89.0
gap4-2	644	o	o	o	o	o	o	o	o	o	o	0	10.3	96.1
gap4-3	673	o	o	o	o	o	o	o	o	o	o	0	0.3	88.2
gap4-4	647	o	o	o	o	o	o	o	o	o	o	0	6.8	93.4
gap4-5	664	o	o	o	o	o	o	o	o	o	o	0	1.7	95.1
gap5-1	563	o	o	o	o	o	o	o	o	o	o	0	2.4	95.2
gap5-2	558	o	o	o	o	o	o	o	o	o	o	0	3.0	94.4
gap5-3	564	o	o	o	o	o	o	o	o	o	o	0	0.9	103.1
gap5-4	568	o	o	o	o	o	o	o	o	o	o	0	0.3	93.7
gap5-5	559	o	o	o	o	o	o	o	o	o	o	0	21.0	115.4
gap6-1	761	o	o	o	o	o	o	o	o	o	o	0	2.1	115.2
gap6-2	759	o	o	o	o	o	o	o	o	o	o	0	15.3	128.7
gap6-3	758	o	o	o	o	o	o	o	o	o	o	0	24.1	134.9
gap6-4	752	o	o	o	o	o	o	o	o	o	o	0	1.2	116.0
gap6-5	747	o	o	o	746	o	746	746	o	746	o	0.05	39.1	155.3
gap7-1	942	o	o	o	o	o	o	o	o	o	o	0	19.1	139.9
gap7-2	949	o	o	o	o	o	o	o	o	o	o	0	5.0	123.9
gap7-3	968	o	o	o	o	o	o	o	o	o	o	0	4.9	125.1
gap7-4	945	o	o	o	o	o	o	o	o	o	o	0	21.2	138.8
gap7-5	951	o	o	o	o	o	o	o	o	o	o	0	2.5	124.0
gap8-1	1133	1132	o	o	1132	1131	1132	1132	1132	1132	1132	0.08	62.5	196.4
gap8-2	1134	o	o	o	o	o	o	o	o	o	o	0	30.6	168.6
gap8-3	1141	1139	1139	1138	1140	o	1139	1140	o	1139	1140	0.12	42.8	177.4
gap8-4	1117	o	1116	1116	o	o	o	o	o	o	o	0.02	76.3	211.4
gap8-5	1127	1126	o	o	1126	1126	o	o	o	o	o	0.03	37.3	168.3
gap9-1	709	o	o	o	o	o	o	o	o	o	o	0	1.1	116.3
gap9-2	717	o	o	o	o	o	o	o	o	o	o	0	42.0	160.0
gap9-3	712	o	o	o	o	o	o	o	o	o	o	0	12.0	127.4
gap9-4	723	o	o	o	o	o	o	o	o	o	o	0	6.6	123.0
gap9-5	706	o	o	o	o	o	o	o	o	o	o	0	7.4	119.9
gap10-1	958	o	o	o	o	o	o	o	o	o	o	0	17.7	152.1
gap10-2	963	o	o	o	o	o	o	o	o	o	o	0	29.4	159.9
gap10-3	960	957	959	958	958	958	958	o	958	958	959	0.18	49.1	188.4
gap10-4	947	945	o	o	o	o	o	o	o	o	o	0.02	27.1	161.0
gap10-5	947	o	o	o	o	o	o	o	o	o	o	0	45.2	176.9
gap11-1	1139	o	o	o	o	o	o	o	o	o	o	0	19.6	171.6
gap11-2	1178	o	o	o	o	o	o	o	1177	o	o	0.01	65.6	225.3
gap11-3	1195	o	o	o	o	o	o	o	o	o	o	0	25.9	202.2
gap11-4	1171	o	o	o	o	o	o	o	o	o	o	0	9.5	164.3
gap11-5	1171	o	o	o	1170	o	o	o	o	o	o	0.01	42.4	201.6
gap12-1	1451	o	o	o	o	o	o	o	1450	o	o	0.01	49.4	263.7
gap12-2	1449	1448	o	o	1448	o	o	o	1448	o	o	0.02	32.7	251.8
gap12-3	1433	1432	1432	o	o	o	1432	1432	1432	o	o	0.03	35.3	265.6
gap12-4	1447	o	o	o	o	o	o	o	1446	o	o	0.01	67.1	293.6
gap12-5	1446	o	o	o	o	o	o	o	o	o	o	0	10.9	227.0

o = optimal solution value.

Table 2. Computational results for 'large-sized' problems

Prob type	Size m	Size n	Best sol'n in each of the 10 trials										Best Overall sol'n	Avg. Best Sol'n time	Avg. Exec. time
A	5	100	o	o	o	o	o	o	o	o	o	o	1698	0.5	435.0
		200	o	o	o	o	o	o	o	o	o	o	3235	0.3	898.4
	10	100	o	o	o	o	o	o	o	o	o	o	1360	0.3	409.4
		200	o	o	o	o	o	o	o	o	o	o	2623	17.0	1213.0
	20	100	o	o	o	o	o	o	o	o	o	o	1158	0.4	737.0
		200	o	o	o	o	o	o	o	o	o	o	2339	43.4	1687.8
B	5	100	1847	b	1847	1849	1846	1848	1852	1859	1860	b	1843	126.9	288.2
		200	3563	3570	3563	3559	3566	3555	3567	b	3565	3566	3553	439.5	790.0
	10	100	b	b	b	1409	b	b	1409	1409	1409	1409	1407	30.1	276.0
		200	2842	2838	2833	2836	2835	2836	2844	2835	b	2837	2831	608.4	1027.4
	20	100	1168	1167	1168	b	1167	1167	b	b	1167	b	1166	191.5	617.3
		200	2341	2341	2344	2341	2341	2341	2343	2341	2341	b	2340	518.5	1323.5
C	5	100	b	1938	1940	1939	1940	1943	1942	b	1942	1937	1931	139.1	302.4
		200	b	3461	3463	3461	3465	3460	3466	3468	3467	3470	3458	531.2	810.1
	10	100	1409	b	b	b	1405	b	1412	1407	1412	b	1403	170.6	394.2
		200	2822	2821	2820	2818	2821	2826	2822	2816	b	2815	2814	628.6	1046.0
	20	100	1247	1254	1255	1249	1247	1247	1251	1249	1251	b	1244	279.9	669.3
		200	2401	b	2406	2410	2402	2409	2404	2412	2409	2408	2397	1095.9	1792.3
D	5	100	6379	6406	6409	6397	6415	b	6388	6391	6406	6384	6373	369.9	530.3
		200	12869	12825	b	12801	12826	12816	12827	12835	12843	12823	12796	1665.9	1942.8
	10	100	6438	6431	6436	6431	6476	6417	6443	6418	b	6407	6379	870.2	1094.7
		200	12603	12641	12638	12654	12605	12633	b	12632	12615	12648	12601	2768.7	3189.6
	20	100	6327	6293	b	6332	6308	6314	6273	6324	6308	6337	6269	1746.1	2126.1
		200	12532	12483	12535	b	12552	12567	12516	12567	12510	12567	12452	4878.4	5565.1

o = optimal solution value.
b = best overall solution value.

- The number of agents m is set to 5, 8 and 10 and the ratio ρ of the number of jobs to the number of agents $\left(\rho = \frac{n}{m}\right)$ is set to 3, 4, 5 and 6 to determine the number of jobs. Five problems are generated for each agent/job combination, giving a total of 60 problems. We have indexed each agent/job combination as follows:

Name	gap1	gap2	gap3	gap4	gap5	gap6	gap7	gap8	gap9	gap10	gap11	gap12
m,n	5,15	5,20	5,25	5,30	8,24	8,32	8,40	8,48	10,30	10,40	10,50	10,60

- The resource requirement r_{ij} are integers from the uniform distribution $U(5, 25)$, the cost

Table 3. Average percentage deviation (σ) from optimal of existing algorithms

Prob set	MTH	FJVBB	FSA	MTBB	SPH	LT1FA	RSSA	TS6	TS1	GA_a	GA_b
gap1	5.43	0.00	0.00	0.00	0.08	1.74	0.00	0.00	0.00	0.00	0.00
gap2	5.02	0.00	0.19	0.00	0.11	0.89	0.00	0.24	0.10	0.01	0.00
gap3	2.14	0.00	0.00	0.00	0.09	1.26	0.00	0.03	0.00	0.01	0.00
gap4	2.35	0.83	0.06	0.18	0.04	0.72	0.00	0.03	0.03	0.03	0.00
gap5	2.63	0.07	0.11	0.00	0.35	1.42	0.00	0.04	0.00	0.10	0.00
gap6	1.67	0.58	0.85	0.52	0.15	0.82	0.05	0.00	0.03	0.08	0.01
gap7	2.02	1.58	0.99	1.32	0.00	1.22	0.02	0.02	0.00	0.08	0.00
gap8	2.45	2.48	0.41	1.32	0.23	1.13	0.10	0.14	0.09	0.33	0.05
gap9	2.18	0.61	1.46	1.06	0.12	1.48	0.08	0.06	0.06	0.17	0.00
gap10	1.75	1.29	1.72	1.15	0.25	1.19	0.14	0.15	0.08	0.27	0.04
gap11	1.78	1.32	1.10	2.01	0.00	1.17	0.05	0.02	0.02	0.20	0.00
gap12	1.37	1.37	1.68	1.55	0.10	0.81	0.11	0.07	0.04	0.17	0.01
Avg % σ_{all}	n/a	n/a	n/a	n/a	n/a	1.15	0.21	0.10	0.07	0.12	0.01
Avg % σ_{best}	2.56	0.84	0.72	0.78	0.13	1.15	0.04	0.06	0.03	0.02	0.00
# Optimal	0	26	n/a	24	40	3	39	40	45	51	60

n/a = not available.
MTH: Martello and Toth [13], constructive heuristic; FJVBB: Fisher, Jaikumar and Van Wassenhove [10], branch-and-bound procedure with an upper CPU limit; FSA: Cattrysse [5], fixing simulated annealing algorithm;
MTBB: Martello and Toth [14], branch-and-bound procedure with an upper CPU limit; SPH: Cattrysse, Salomon and Van Wassenhove [6], set partitioning heuristic;
LT1FA: Osman [2], long term descent with 1-interchange mechanism and first-admissible selection;
RSSA: Osman [2], hybrid simulated annealing/tabu search;
TS6: Osman [2], long term tabu search with first-admissible selection;
TS1: Osman [2], long term tabu search with best-admissible selection;
GA_a: GA without the heuristic operator;
GA_b: GA with the heuristic operator.

coefficients c_{ij} are integers from $U(15,25)$ and the capacity of agents $b_i = 0.8 \sum_{j \in J} r_{ij}/m$.

The remaining 24 problems are categorised as 'large-sized' problems. Although large problems have been attempted by several authors in the past [7–9], there has not been a standard set which is available to the research community. Therefore we generate these publicly available 'large-sized' test problems, based on the standard (but not uniquely defined, varying between authors) GAP generation scheme [7,10–12]. These problems consist of four types:

Type A. r_{ij} are integers from U (5,25), c_{ij} are integers from U (10,50) and $b_i = 0.6(n/m)15 + 0.4R$ where $R = \max_{i \in I} \sum_{j \in J, I_j = i} r_{ij}$ *and* $I_j = \min[i | c_{ij} \leq c_{kj}, \forall k \in I]$.

Type B. r_{ij} and c_{ij} are the same as Type A and b_i is set to 70% of the value given in Type A.

Type C. r_{ij} and c_{ij} are the same as Type A and $b_i = 0.8 \Sigma_{j \in J} r_{ij}/m$.

Type D. r_{ij} are integers from U (1,100), $c_{ij} = 111 - r_{ij} + e$ where e are integers from U (-10, 10) and $b_i = 0.8 \Sigma_{j \in J} r_{ij}/m$.

For each problem type, we generate one problem for each agent/job combination ($m = 5$, 10, 20 and $n = 100$, 200), giving a total of 24 problems.

The 60 'small-sized' problems have known optimal solutions which were obtained by applying a branch-and-bound procedure requiring computation times ranging from a few seconds to a few hours [5]. Note that the optimal solution values are given when these problems are solved as maximisation problems. Since we solve the GAP as a minimisation problem, we changed the sign of the costs in the original data such that all costs become negative and the problems are solved as minimisation problems. The absolute values of the final objective function values are then used to compare with the given optimal solution values. For the 24 'large-size' problems the optimal solutions are not known, except for type A problems for which we were able to obtain the optimal solutions by using the CPLEX general purpose mixed integer solver.

In our computational study, 10 trials of the GA heuristic were performed for each of the 84 problems. Each trial terminated when $M = 500000$ non-duplicate children have been generated without improving the best solution found and the population size N is set to 100. We allow this many solutions to be generated because we are more interested in producing good quality solutions and less concerned about the computational expense. For each problem, we report the known optimal solution value, the best solution value for each of the 10 trials, the average percentage deviation (σ) from optimal ($\sigma = \sum_{i=1}^{10} \frac{S_o - S_i}{10 S_o} \times 100\%$ where S_i is the best solution value of the i-th trial and S_o is the optimal solution value), the average best-solution time (in CPU seconds) which is the time that the GA takes to first reach the final best solution and the average execution time (in CPU seconds) which is the total time the GA takes before termination. The results for the 'small-sized' and 'large-sized' problems are shown in Tables 1 and 2 respectively. Examining Table 1, we observe that the GA performs very well and finds the optimal solution in at least one trial for *all* 60 problems. For those few trials in which the GA failed to reach the optimal solution, all solution values are very close to optimality. The average execution times are all within a reasonable time of 300 CPU seconds. The average best-solution time also falls well below the average execution time. This indicates that similar results may be obtained by using a more restrictive stopping criteria. We would comment here that our experience has been that whilst GA's are usually not the quickest heuristic available for solving a problem, they are capable of producing extremely good quality results within a reasonable computation time.

In Table 2, we give our heuristic solution values for the 'large-sized' problems as a benchmark for future comparison with other heuristic algorithms. For the Type A problems, our best solution values match the optimal values obtained by the CPLEX mixed integer solver. The average best-solution times, similar to the solution times taken by CPLEX except in two instances, are only a fraction of a CPU second, thus confirming the (known) fact that Type A problems are very easy to solve. For Type B, C and D problems, the average best-solution times are much larger. For the Type D problems in particular, the average number of non-duplicate children generated is approximately 3×10^6 before termination, indicating that the solution quality in Type D improves more slowly during the course of a trial than in

the other problem types.

In Table 3, we compare the performance of our GA heuristic with other existing heuristic algorithms in terms of the average percentage deviation (σ) for each problem set, the average percentage deviation for all problems (σ_{all}), the average percentage deviation of the best solutions (σ_{best}) and the number of optimal solutions obtained (out of a total of 60) for each of the 12 'small-sized' problem sets. This summary data for the existing heuristic algorithms (except for our GA) is from Osman [2]. We also compare the results for the GA without (GA_a) and with (GA_b, as in Table 1) the heuristic operator. It can be seen that the performance of our GA heuristic is better than all other heuristic algorithms in terms of the solution quality, whilst the GA with the heuristic operator performs better than that without it. Although the computational time for GA_b is on average 30% more than that for GA_a when the same number of children are generated, this extra computational effort is justified considering the significant improvement that GA_b gives over GA_a. The computational times for other algorithms are not given here since it is difficult to compare different codes (possibly programmed with varying degrees of efficiency), CPU times on different hardware platforms and algorithms with different stopping criteria. But generally speaking the CPU times allowed for these other algorithms before termination are shorter than those given for the GA. Whether these other algorithms would have been able to attain solutions of equal, or better, quality than the GA if run on the same CPU for the same time is simply not known.

5. CONCLUSIONS

We have presented a heuristic for the generalised assignment problem based on a genetic algorithm. Our GA heuristic features a non-binary representation that automatically satisfies the job assignment constraints, a fitness-unfitness pair evaluation function and a heuristic operator which helps to improve the cost and feasibility of a solution. Although the GA without the heuristic operator is capable of producing good results, the results can be greatly improved if the heuristic operator is used and hence the additional computational effort involved is justified. Computational results show that our GA heuristic is able to generate optimal solutions for many 'small-sized' problems and the average percentage deviation from optimality for these problems is only 0.01%. This performance is achieved within reasonable computation times.

Comparing our GA heuristic with other existing heuristic algorithms, the GA is superior in terms of solution quality, although the computational times for some heuristics may be significantly shorter than those given for the GA. Further, 'large-sized' problems are generated and tested using the GA heuristic. We have made these test problems publicly available to aid future heuristic development for the GAP.

REFERENCES

1. Cattrysse, D. and Van Wassenhove, L. N., A survey of algorithms for the generalized assignment problem. *Eur. J. Oper. Res.*, 1992, **60**, 260–272.
2. Osman, I. H., Heuristics for the generalised assignment problem: simulated annealing and tabu search approaches. *OR Spektrum*, 1995, **17**, 211–225.
3. Beasley, J. E. and Chu, P. C., *A Genetic Algorithm for the Set Covering Problem. Eur. J. Oper. Res.* (in press).
4. Chu, P. C. and Beasley, J. E., A genetic algorithm for the set partitioning problem. Working paper. The Management School, Imperial College, London SW7 2AZ, England, 1995.
5. Cattrysse, D., Set partitioning approaches to combinatorial optimization problems. Ph.D. thesis, Katholieke Universiteit Leuven, Centrum Industrieel Beleid, Belgium, 1990.
6. Cattrysse, D., Salomon, M. and Van Wassenhove, L. N., A set partitioning heuristic for the generalized assignment problem. *Eur. J. Oper. Res.* , 1994, **72**, 167–174.
7. Amini, M. M. and Racer, M., A rigorous computational comparison of alternative solution methods for the generalized assignment problem. *Mgmt Sci.* , 1994, **40**, 868–890.
8. Klastorin, T. D., An effective subgradient algorithm for the generalized assignment problem. *Comp. Oper. Res.* , 1979, **6**, 155–164.
9. Trick, M., A linear relaxation heuristic for the generalised assignment problem. *Naval Res. Logist.* , 1992, **39**, 137–151.
10. Fisher, M. L., Jaikumar, R. and Van Wassenhove, L. N., A multiplier adjustment method for the generalized assignment problem. *Mgmt Sci.* , 1986, **32**, 1095–1103.
11. Martello, S. and Toth, P., Linear assignment problems, *Annals of Discrete Mathematics*, 1987, **31**, 259–282.
12. Ross, G. T. and Soland, R. M., A branch and bound algorithm for the generalized assignment problem. *Math. Prog.* , 1975, **8**, 91–103.
13. Martello, S. and Toth, P., An algorithm for the generalized assignment problem. In *Operational Research '81*, ed. J. P. Brans. North-Holland, 1981, pp.589–603.
14. Martello, S. and Toth, P. *Knapsack Problems: Algorithms and Computer Implementations*. Wiley, New York, 1990.