Pergamon

0305-0548(93)E0020-T

*Computers Ops Res.* Vol. 22, No. 1, pp. 73–83, 1995
Copyright © 1994 Elsevier Science Ltd
Printed in Great Britain. All rights reserved
0305-0548/95 $7.00+0.00

# A GENETIC APPROACH TO THE QUADRATIC ASSIGNMENT PROBLEM

DAVID M. TATE† and ALICE E. SMITH‡§

Department of Industrial Engineering, 1048 Benedum Hall, University of Pittsburgh, Pittsburgh, PA 15261, U.S.A.

**Scope and Purpose**—There are a vast number of practical design and resource-allocation problems, in many different fields, where the decision to be made is a matching (or assignment) of items in one set to items in another, disjoint set. If the costs associated are simply constants for each possible pairing, this is the classical "Assignment Problem", for which good algorithms have been known for more than a century. However, if the cost structure is more complex, so that the cost of a given assignment depends on two-way or higher-order interactions between pairings, difficult combinatorial problems result. The quadratic assignment problem (QAP) is perhaps the simplest in structure of these difficult problems. In QAP, a constant cost is associated with simultaneouly making two particular assignments. Such cost structures arise, for example, in facility location problems, where the cost of locating facility $i$ at site $j$ and facility $k$ at site $l$ is a function of the distance between the two sites $j$ and $l$, and the degree of interaction between the two facilities $i$ and $k$. Genetic algorithms (GA) are a family of parallel, randomized-search optimization heuristics which are based on the biological process of natural selection. They have proven to be most effective on non-convex optimization problems for which it is relatively easy to assess the quality of a given feasible solution, but difficult to systematically improve solutions by deterministic iterative methods. Most NP-complete combinatorial problems fall into this category. Since QAP is the nonlinear assignment problem with the most "special structure", it is more likely to yield good solutions to clever deterministic heuristics which take advantage of that structure. Conversely, if GAs can be shown to perform competitively on QAP, this gives us good reason to believe that extending genetic algorithms to the many more complex nonlinear assignment problems found in VLSI design, facility layout, and location problems may yield better results than deterministic heuristics can provide for these less-structured problems. In this paper, we present the results of an investigation of a particular GA for QAP, and discuss the potential of GAs for more complex nonlinear assignment problems. We show that the GA performed consistently equal to or better than previously known heuristics without undue computational overhead. We conclude with some more general comments on the design and implementation of GAs, motivated by our results for QAP.

**Abstract**—The quadratic assignment problem (QAP) is a well-known combinatorial optimization problem with a wide variety of practical applications. Although many heuristics and semi-enumerative procedures for QAP have been proposed, no dominant algorithm has emerged. In this paper, we describe a genetic algorithm (GA) approach to QAP. Genetic algorithms are a class of randomized parallel search heuristics which emulate biological natural selection on a population of feasible solutions. We present computational results which show that this GA approach finds solutions competitive with those of the best previously-known heuristics, and argue that genetic algorithms provide a particularly robust method for QAP and its more complex extensions.

## 1. INTRODUCTION

The quadratic assignment problem (QAP) is a well-known classical combinatorial optimization problem, which can be described as follows. We are given a set of $n$ distinct **objects** $1, 2, \ldots, n$, which are to be placed uniquely in $n$ or more distinct **sites**. Associated with each pair of objects $(i, j)$ is an interaction (or **traffic**) intensity, $T(i, j)$. Associated with each pair of sites $(S_k, S_l)$ is a unit traffic cost (often referred to as the **distance** between $S_k$ and $S_l$), $D(S_k, S_l)$. We may also specify a

†David M. Tate is an Assistant Professor in the Department of Industrial Engineering at the University of Pittsburgh. His research interests include heuristic combinatorial optimization, characterization and control of stochastic systems with due-dates, and discrete-event simulation methodology.

‡Alice E. Smith is an Assistant Professor in the Department of Industrial Engineering at the University of Pittsburgh. She received a Ph.D. in Engineering Management from the University of Missouri at Rolla, an M.B.A. from Saint Louis University and a B.S.C.E. from Rice University. Her research interests are computational intelligence in manufacturing, including neural networks, genetic algorithms and fuzzy systems. Her articles have appeared in *International Journal of Production Research, Journal of Intelligent Manufacturing, International Journal of Advanced Manufacturing Technology*, and *The Engineering Economist*.

§To whom all correspondence should be addressed.

fixed cost $F(i, S_j)$ associated with the placement of object $i$ in site $S_j$. If we let $A(i)$ denote the site to which object $i$ is assigned in a particular feasible assignment $A$, the total cost of assignment $A$ is given by

$$\text{Cost}(A) = \sum_{\text{objects } i} F(i, A(i)) + \sum_{\text{objects } i} \sum_{\text{sites } j} [T(i, j) \cdot D(A(i), A(j))].$$

The distance matrix $D$ need not be symmetric; the fixed costs may be zero.

There are numerous examples of applications of this problem. If we define the sites to be particular locations on a plane, our objects to be facilities which will interact, the distances to be unit directed travel costs between locations, and the traffic intensities to be the pairwise amount of travel required between facilities, then we get the well known facility layout problem. If we choose our sites to be pre-specified points on a printed circuit card, our objects to be required components, our traffic intensities to be the number of wires required between components, and our distances to be the inter-site wire lengths required, we get a component placement problem. Since Koopmans and Beckmann [18] first used a QAP formulation in the context of Facility Layout Planning, more than fifty articles have appeared in the OR literature proposing heuristics procedures and enumerative schemes for QAP. Among the more influential of these have been Hillier and Connors [11], Beghin-Picavet and Hansen [2], Burkard and Bonniger [3], Bazaraa and Kirca [1], Picone and Wilhelm [20], Wilhelm and Ward [23], Heragu and Kusiak [10], and Kaku *et al.* [17].

QAP is known to be NP-complete [8]. We describe below an approach for finding consistently good solutions to instances of QAP using a genetic algorithm (GA) framework, and show that this approach yields solutions comparable to the best previously-known heuristics for QAP when applied to test problems from the literature. We argue that this method is not overly costly in terms of computation effort, and is exceptionally robust both with respect to variations in the objective function, constraints of the problem, and the details of the GA implementation. We conclude with some general remarks about the implementation of GAs, motivated by our observations in this study.

## 2. OVERVIEW OF GENETIC ALGORITHMS

The phrase "genetic algorithms" denotes a family of parallel, randomized-search optimization heuristics which share the following features:

(a) One or more "populations" of feasible solutions.
(b) A mechanism for generating new feasible solutions by combining features from multiple previously-known solutions.
(c) A mechanism for generating a new feasible solution by a random perturbation of a single previously-known solution.
(d) A mechanism for selecting individual solutions from the population(s), giving preference to those with better objective function values.
(e) A mechanism for removing solutions from the population(s).

The motivation for GA is the biological paradigm of natural selection as first articulated by Holland [13]. We thus refer to the mechanism of (b) above as a **breeding** or **reproduction** mechanism, that of (c) as a **mutation** mechanism, that of (d) as a **selection** mechanism and that of (e) as a **culling** mechanism. The basic flow of any GA implementation is thus given by the following:

```
1. Create an initial population or sub-populations of solutions, usually feasible and
random.
2. Repeat {
    2.1 Select parents.
    2.2 Breed offspring, and add them to the population.
    2.3 Mutate certain members of the current population.
    2.4 Cull certain members of the current population.
}
3. Until (some termination criterion is met).
```

There is a great deal of flexibility in the choice of how (and when) to select, breed, mutate and cull individuals in the population. For example, we might choose to replace all members of the current population with bred offspring at every generation of the algorithm. We might decide to incorporate the possibility of mutation within our breeding algorithm, or to mutate individuals separately. We might select parents according to their absolute **fitness** (the objective function value), or their rank in the current population, or some other criterion. We might maintain one large population, or several smaller, parallel sub-populations.

Perhaps more importantly, there are any number of different possible breeding and mutation schemes for a given problem type, which will in turn depend on how we choose to **encode** solutions. Typically, breeding and mutation algorithms operate on solutions only through this encoding, so that a particular choice of encoding imposes a topology on the response surface of the underlying optimization problem. This can have profound effects on the success of a GA approach to a given problem family; a particularly clever encoding can restrict the necessary search area to be an extremely small subset of all feasible solutions, while at the same time allowing computationally efficient breeding and mutation of new feasible solutions. We will elaborate on this observation in our concluding remarks.

Since Holland's seminal work, GAs have been applied to such optimization problems as job shop scheduling by Davis [6], sizing of communications networks by Davis and Coombs [7], Flexible Manufacturing System scheduling [15], and multiprocessor task scheduling [14,21]. A component placement problem, related to the QAP, has been addressed using genetic algorithms by Cohoon and Paris [5] and Cohoon *et al.* [4].

## 3. THE GENETIC ALGORITHM APPROACH TO THE QUADRATIC ASSIGNMENT PROBLEM

### *3.1. Encoding, selection and reproduction*

We applied genetic search to the QAP. For the test problems considered, the possible sites for assignment are a rectangular lattice of locations in the plane, with the inter-site distance computed either by Euclidean ($L_2$) norm or Manhattan ($L_1$) norm. Sites were ordered in boustrophedon sequence by rows, establishing a one-to-one correspondence between sequences of $\{1, 2, \ldots, n\}$ and feasible assignments. The number of rows and columns were pre-specified and fixed, and unused sites were assigned "dummy" objects with no traffic in or out. Each population member was a feasible solution (i.e. each object is assigned a unique site).

Since we did not use a traditional binary encoding for solution strings, we were able to design reproduction and mutation functions which operate directly on the solution sequence. Mutation and reproduction were completely independent, unlike most GA implementations where mutation only takes place on a newly formed child. We selected solutions for mutation from the entire population with uniform probability. Mutation took the form of selecting two sites at random and reversing the order of all sites within the subsequence bounded by the two selected. For selected sites either adjacent, or one space apart in the boustrophedon sequence, this reduces to pairwise interchange.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Selected String: | B | C | F | A | G | E | H | D |
| Random Mutation Sites: | | x | | | x | | | |
| Mutated String: | B | G | A | F | C | E | H | D |

Selection for reproduction was performed with a bias towards choosing the better solutions in the current population. We chose not to select proportional to objective function value, as is done with the traditional biased roulette wheel approach. Instead, we selected parent strings by choosing a uniform random number between 1 and $\sqrt{m}$ (where $m$ is the population size), then squaring it. The result was truncated and taken to be the rank of the selected parent (where string zero is the fittest string in the population). This approach can be generalized to give an arbitrary degree of preference or even varied dynamically during a given run by changing the power of the root of $m$ during subsequent generations.

We could not adopt the standard crossover since it would not maintain feasibility of our non-binary coding. Our reproduction scheme was as follows. Any object assigned to the same site in both parents occupies that site in the offspring. For the remaining sites, one or the other objects assigned to that site in one parent is chosen at random, working left to right through the sequence of sites. Any unassigned objects are then matched with the remaining unassigned sites. This reproduction scheme is notable for its ability to generate multiple potentially different children from the same two parents. An example is shown below.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Parent 1: | D | C | A | H | F | E | B | G |
| Parent 2: | F | A | E | H | B | D | C | G |
| Common Locations: | — | — | — | H | — | — | — | G |
| Random Choice: | D | A | E | H | F | — | B | G |
| Leftover: | C | | | | | | | |
| Finished Child: | D | A | E | H | F | C | B | G |

For each generation, the number of children to be produced was selected in advance, as was the probability of a given individual generating a mutant. A mutant was distinct from the individual which spawned it, and both had opportunities to survive. As children and mutants were created, the strings with worst fitness were culled to keep a constant population size. Thus, a new child might be deleted from the population in the same generation as its creation. To maintain enough diversity, mutants were inserted into the population after members of the current generation had been culled. In this way, mutants, no matter what their fitness, were assured to survive at least one generation and be available for reproduction selection. The single best solution was always retained unaltered for the next generation to encourage convergence.

## 3.2. *Description of test problems and methodology*

The most well known QAP instances for rectilinear distance are those eight defined in Nugent *et al.* [19], with fifteen subsequent research results summarized in Hundal [16]. These range in size from 5 objects (6 sites) to 30 objects (30 sites), where the traffic matrix is assumed symmetric. For formulations where the number of sites is greater than the number of objects, we defined dummy objects with zero traffic interaction. For the 7 object problem with 9 sites, we used 2 dummy objects which were forced to be adjacent by a large $T_{ij}$ between them. Another special case of the rectilinear QAP is that cited in Cohoon *et al.* [4], consisting of 16 sites and objects with binary traffic costs (i.e. the $T_{ij}$ matrix consists of 1s and 0s only).

The most well known Euclidean distance QAP problem is a 36 site and 34 object computer component placement problem first documented in Steinberg [22]. This problem served as a comparison for at least seven subsequent works, and results are summarized in Hanan and Kurtzberg [9]. The 34 objects are set in a 4 by 9 grid of sites, and we included two dummy departments, not forced together. We note that the original Steinberg paper contains an error in the Connection Matrix (his Fig. 1), and we assume that the correct entry for the number of connections between E28 and E29 is 10, rather than 0, as one of Steinberg's entries indicates. As another test of a Euclidean distance problem, we formulated a 25 site and object problem where we would know the optimum *a priori*. This problem is shown in Fig. 1.

We ran our GA on each problem multiple times. This was for two reasons. First, GAs are stochastic and therefore yield different searches, and potentially different results, for each random number seed used. We arbitrarily selected ten seeds and used those same seeds to create ten genetic runs for each problem. These ten runs tested the dependence of the method on the seed selected, both in terms of solution quality and search effort. The second aspect of multiple runs was to test the best mix of reproduction and mutation parameters for our particular genetic algorithm. Since we had not worked with this implementation before we had no basis for choosing the best mix *a priori*. We ran each problem and each random number seed with the following three mixes, listed in order of decreasing randomness: 25% children and 75% probability of mutation, 50% children and 50% probability of mutation, and 75% children and 25% probability of mutation. The percent children refers to the number of children created each generation as a percent of population. The probability of mutation is the probability that any one solution will undergo mutation during a generation, so that a 25% mutation probability will yield an expected value of 25 mutants per generation for a population of 100. Other than the three parameter mixes and the ten random number seeds, the GA algorithm was unchanged for each run.

The experiment totaled 30 runs for each of the eleven problems. Each run consisted of a fixed

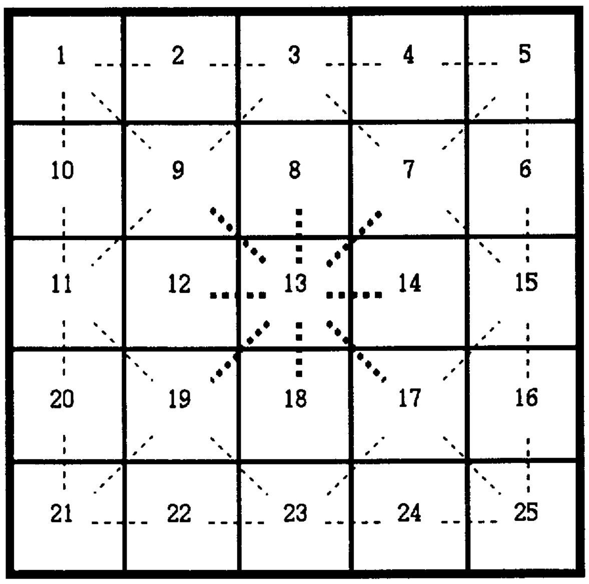

Fig. 1. Our 25 site Euclidean problem. Where heavy line traffic = 50 and light line traffic = 10, and each site is a unit square.

Table 1. Objective function values of solutions found

| | 25% Child/75% Mut. | | | 50% Child/50% Mut. | | | 75% Child/25% Mut. | | |
|---|---|---|---|---|---|---|---|---|---|
| Problem | Best | Mean | C. Var. | Best | Mean | C. Var. | Best | Mean | C. Var. |
| 5/6 R | 25* | 25* | 0.0 | 25* | 25* | 0.0 | 25* | 25* | 0.0 |
| 6/6 R | 43* | 43* | 0.0 | 43* | 43* | 0.0 | 43* | 43* | 0.0 |
| 7/9 R | 74* | 74* | 0.0 | 74* | 74* | 0.0 | 74* | 74.7 | 0.008 |
| 8/8 R | 107* | 107* | 0.0 | 107* | 107* | 0.0 | 107* | 107.5 | 0.01 |
| 12/12 R | 289* | 291.4 | 0.007 | 289* | 294 | 0.013 | 289* | 297.9 | 0.018 |
| 15/15 R | 575* | 585 | 0.011 | 576 | 589.5 | 0.017 | 581 | 596.5 | 0.015 |
| 20/20 R | 1304 | 1321.7 | 0.008 | 1309 | 1320.1 | 0.007 | 1299 | 1335.7 | 0.013 |
| 30/30 R | 3092 | 3152.7 | 0.011 | 3092 | 3160 | 0.013 | 3113 | 3181.6 | 0.011 |
| 16/16 R | 48* | 54.4 | 0.152 | 48* | 58 | 0.150 | 48* | 61.8 | 0.104 |
| 25/25 E | 1625.1 | 1768.3 | 0.036 | 1624.6* | 1768.4 | 0.042 | 1758.8 | 1828.4 | 0.031 |
| 34/36 E | 4296.0 | 4473.1 | 0.023 | 4271.5 | 4382.2 | 0.030 | 4385.6 | 4490.5 | 0.016 |

*Equal to, or better than, the best previously published solution.

population size of 100 solutions with a randomly generated initial feasible population of 100 individuals. The search was allowed to continue until either the optimum had been reached (if it was known) or 2000 generations had passed. The best solution yet found and total number of solutions generated were recorded every 20 generations. The number of solutions generated was not adjusted for duplicates, and therefore represents an overestimate of the number of unique solutions investigated.

## *3.3. Results*

Table 1 shows the objective function values of the best and mean solutions and the coefficient of variation over the ten solutions for each parameter mix. Asterisks denote best known solution values, i.e. equal to or better than the best previously published solution. Solutions reported without asterisks are greater than the best previously published solutions, as is more fully detailed in Table 3. Problems are denoted by first the number of objects, then the number of sites, and then the distance measure [rectilinear (R) or Euclidean (E)].

For each test problem, we show in the Appendix one configuration of the best solution found by this genetic algorithm approach with its objective function value. We found overall that our best mix of reproduction and mutation was the most stochastic one with 25% children and 75% probability of mutation, however results are not dramatically different for the 50% children and 50% mutation formulation. The 75% children and 25% mutation was consistently worst in terms

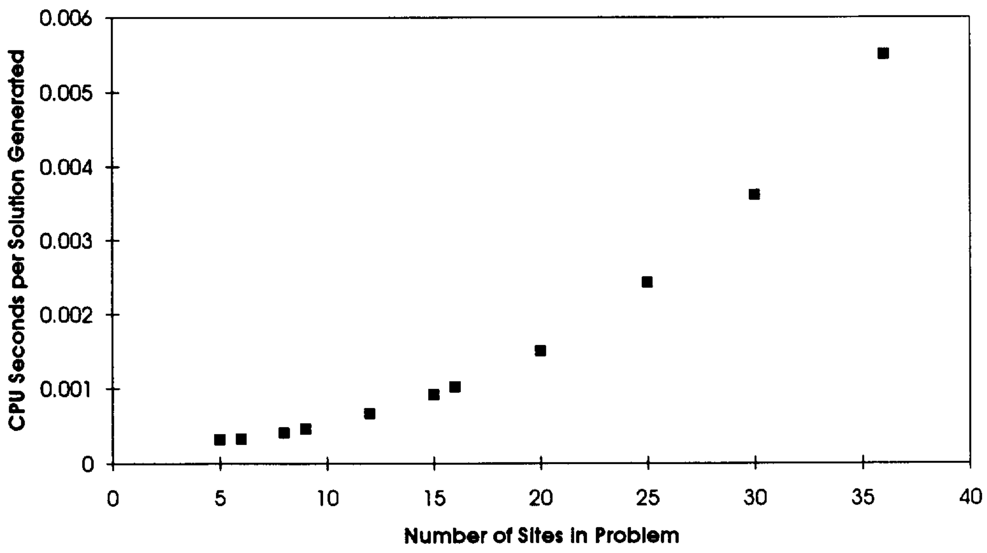

Fig. 2. CPU time per solution generated for each problem.

Table 2. Number of solutions explored before finding the best

| | 25% Child/75% Mut. | | | 50% Child/50% Mut. | | | 75% Child/25% Mut. | | |
|---|---|---|---|---|---|---|---|---|---|
| Problem | Quick | Mean | C. Var. | Quick | Mean | C. Var. | Quick | Mean | C. Var. |
| 5/6 R | 269 | 277.4 | 0.018 | 243 | 290 | 0.176 | 203 | 344.7 | 0.696 |
| 6/6 R | 272 | 399.8 | 0.306 | 194 | 349.7 | 0.375 | 174 | 706.2 | 0.810 |
| 7/9 R | 384 | 1153.7 | 0.986 | 192 | 784.8 | 0.726 | 430 | 1382 | 0.690 |
| 8/8 R | 384 | 1342.9 | 0.588 | 214 | 6854.3 | 1.946 | 236 | 2328.1 | 1.117 |
| 12/12 R | 2097 | 18,892 | 1.351 | 2069 | 7100 | 0.653 | 2084 | 13,323 | 2.037 |
| 15/15 R | 4080 | 39,099 | 1.124 | 2091 | 9481.2 | 1.105 | 2082 | 21,907 | 1.723 |
| 20/20 R | 8150 | 75,084 | 0.920 | 4093 | 70,249 | 0.944 | 2099 | 33,105 | 1.489 |
| 30/30 R | 13,633 | 110,010 | 0.581 | 13,211 | 84,499 | 0.686 | 92,185 | 148,151 | 0.273 |
| 16/16 R | 4107 | 14,707 | 1.21 | 2126 | 13,493 | 0.910 | 2068 | 25,915 | 1.164 |
| 25/25 E | 20,118 | 72,262 | 0.546 | 8147 | 56,496 | 1.040 | 26,093 | 83,330 | 0.776 |
| 34/36 E | 32,125 | 123,650 | 0.410 | 38,059 | 145,674 | 0.333 | 26,093 | 122,331 | 0.446 |

of average solution quality. The success of our high mutation rate is probably due to the conservative nature of both our reproduction and mutation, i.e. much of the structure of the original string is generally left intact through these operations. Differences existed for different random number seeds, but solution quality was within a few percentage points, as evidenced by the extremely low coefficient of variation over the 10 seeds (except for the 16 department problem with 0 or 1 flows). In stochastic search, an approach which is robust with respect to solution quality is a paramount concern. Our GA not only performed well, it performed well consistently. As with any stochastic search method, multiple runs are desirable if the problem is large or complex, and near-optimality is required.

For NP-complete problems of medium to large size, the search effort required to attain a given level of quality is extremely important. In Fig. 2 we show the CPU time per solution generated for each problem size on an Iris 4D25 workstation running under UNIX. (Our GA was coded in ANSI C, and we did not try to optimize for computational performance.) We generated approx. 200,000 solutions per GA run. Since effort increases primarily for the distance calculation, we estimate the increase in computational effort per solution generated is quadratic with the number of sites.

Since CPU time is highly dependent on hardware and software, we also report the number of solutions evaluated to find the best solution among the 2000 generations. Table 2 shows the least number of solutions searched to find a single run's best solution (the column titled "Quick"), the mean number of solutions searched over the ten seeds, and the coefficient of variation of solutions searched over the ten seeds. Since the coefficient of variation among runs is large, we cannot draw conclusions about the number of solutions needed to be generated for a given problem size from

Table 3. Our results as a percent over best known solutions

| | 25% C/75% M | | 50% C/50% M | | 75% C/25% M | |
|---|---|---|---|---|---|---|
| Problem | Best | Worst | Best | Worst | Best | Worst |
| 5/6 R | 0 | 0 | 0 | 0 | 0 | 0 |
| 6/6 R | 0 | 0 | 0 | 0 | 0 | 0 |
| 7/9 R | 0 | 0 | 0 | 0 | 0 | 2.299 |
| 8/8 R | 0 | 0 | 0 | 0 | 0 | 2.804 |
| 12/12 R | 0 | 1.384 | 0 | 4.152 | 0 | 5.536 |
| 15/15 R | 0 | 3.130 | 0.174 | 6.435 | 1.043 | 6.087 |
| 20/20 R | 0.696 | 3.403 | 1.237 | 3.403 | 0.464 | 5.027 |
| 30/30 R | 0.422 | 4.190 | 0.422 | 5.391 | 1.104 | 4.774 |
| 16/16 R | 0 | 33.33 | 0 | 41.67 | 0 | 41.67 |
| 25/25 E | 0 | 15.075 | −0.033* | 16.606 | 8.228 | 18.979 |
| 34/36 E | 1.752 | 10.602 | 1.173 | 10.614 | 3.863 | 8.942 |

*The solution found bested our *a priori* "optimum" solution by an innovative reconfiguration.

our research. We can make two general observations, however. First, the mean number of solutions explored to attain comparable quality solutions generally increases with problem size. Second, the fraction of solution space explored for comparable quality solutions decreases rapidly with problem size.

Another important comparison is to previously published results or known optimum results. This is possible for all of our problems with the exception of the 25 object/site Euclidean problem which we devised and for which we assumed (incorrectly) the pre-formulated solution to be the optimum. Table 3 shows our best found solution over the 10 runs of each problem as a percent above the best known solution. To show the reliability of our approach, Table 3 also shows the worst final solution found as a percent over the best known solution. Our best solutions were generally optimum, or within a few percentage points. Worst solutions were still reasonable, with the notable exception of the 16 object/site rectilinear problem, where a miss carried a high penalty because of the 0/1 flow matrix.

With regard to the quality of our best solutions, for the classical rectilinear problems of Nugent *et al.* one approach's best solution consistently bettered ours. This approach, the terminal sampling procedure of Hitchings and Cottam [12], found better solutions for the 20 and 30 objects/sites problems. The improvement was $<0.25\%$ however, and, at least for the 30 site problem, our mean solution was better than the average of the solutions they report. It is not known whether these reported solutions were representative. They did not report any results other than their best found for the 20 site problem. The only approach which improved in quality of solutions over ours for the 34 object/36 site Euclidean problem was Raymond's Linearization [9]. Our results during evolution for this problem using the three different GA parameter mixes are shown in Fig. 3, along with four other well known approaches (Raymond's Linearization as the $x$ axis or baseline solution, and Hillier's Interchange, Gilmore's $n^4$ Algorithm and Steinberg shown as dashed lines at their best found solution level).

As mentioned earlier, the genetic search found an unexpected best solution for the 25 site Euclidean problem. For the 16 site rectilinear problem from Cohoon *et al.* [4], our GA approach performed considerably better than both the genetic approach and simulated annealing approach tested by those authors. Cohoon *et al.*'s GA is tailored to a more general problem, however, in which potential locations are also variable, and not all assignments are feasible. The only conclusion we draw is that the QAP, as a special case of the placement problem, benefits substantially from a special purpose algorithm.

## 4. INTERPRETATION OF RESULTS

The performance of this GA implementation is very encouraging. A simple mutation and reproduction scheme was able to match the best known heuristics from previous research, without careful selection of mutation rate, population size, or any other specific parameters of the implementation. This suggests, on the one hand, that there is potential for improved performance by a GA with a more sophisticated reproduction or mutation scheme, so that this implementation is not necessarily a "dead end" for this problem class. On the other hand, the apparent robustness

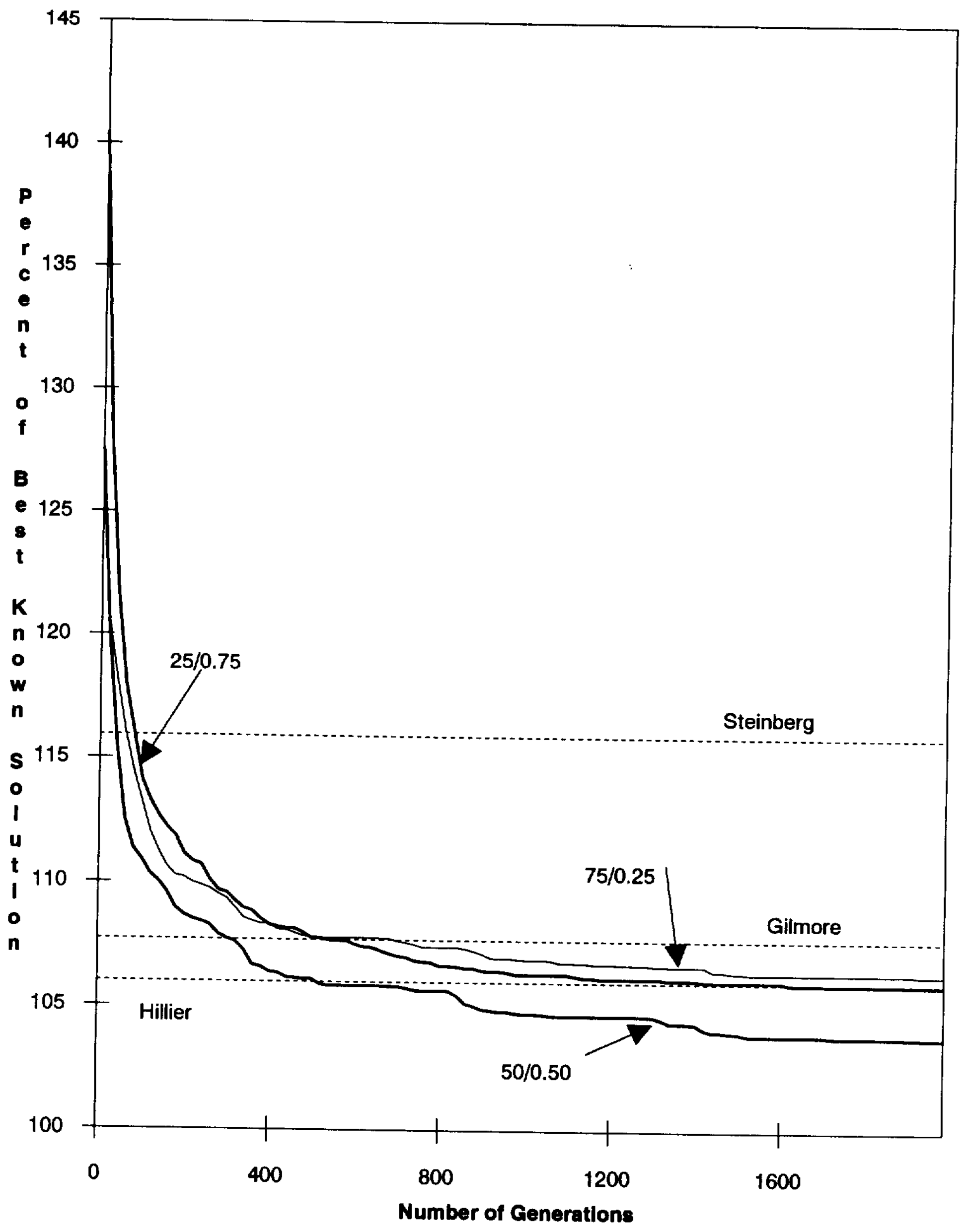

Fig. 3. Our results compared to previous research for the 34 object problem.

of the implementation with regard to initial settings allows us to be optimistic that one can achieve near-optimal solutions quickly without having a fine-tune sensitive parameters of the implementation, or to explore a wide variety of encodings and generation algorithms. The excellent average-case performance of the GA approach across different initial populations and pseudo random number seeds reinforces this impression of robustness.

There are a few important general conclusions that are suggested by the experimental results. The first has to do with the importance of culling. Figure 4 shows a comparison of the average convergence rates of the "best solution found by *k*th generation" for three different sets of runs, using pure mutation (i.e. no reproduction). Each of the three sets generated approx. 200,000 solutions. The only difference among them was in what fraction of the population was culled in each generation. In the case of 100% culling, each generation represents only the mutated "offspring" of the previous generation's individuals. Convergence is slow, and does not seem to be converging toward the best known solutions. At the other extreme, replacing only the bottom 25% of the population in any given generation leads to convergence almost as fast and deep as that of the

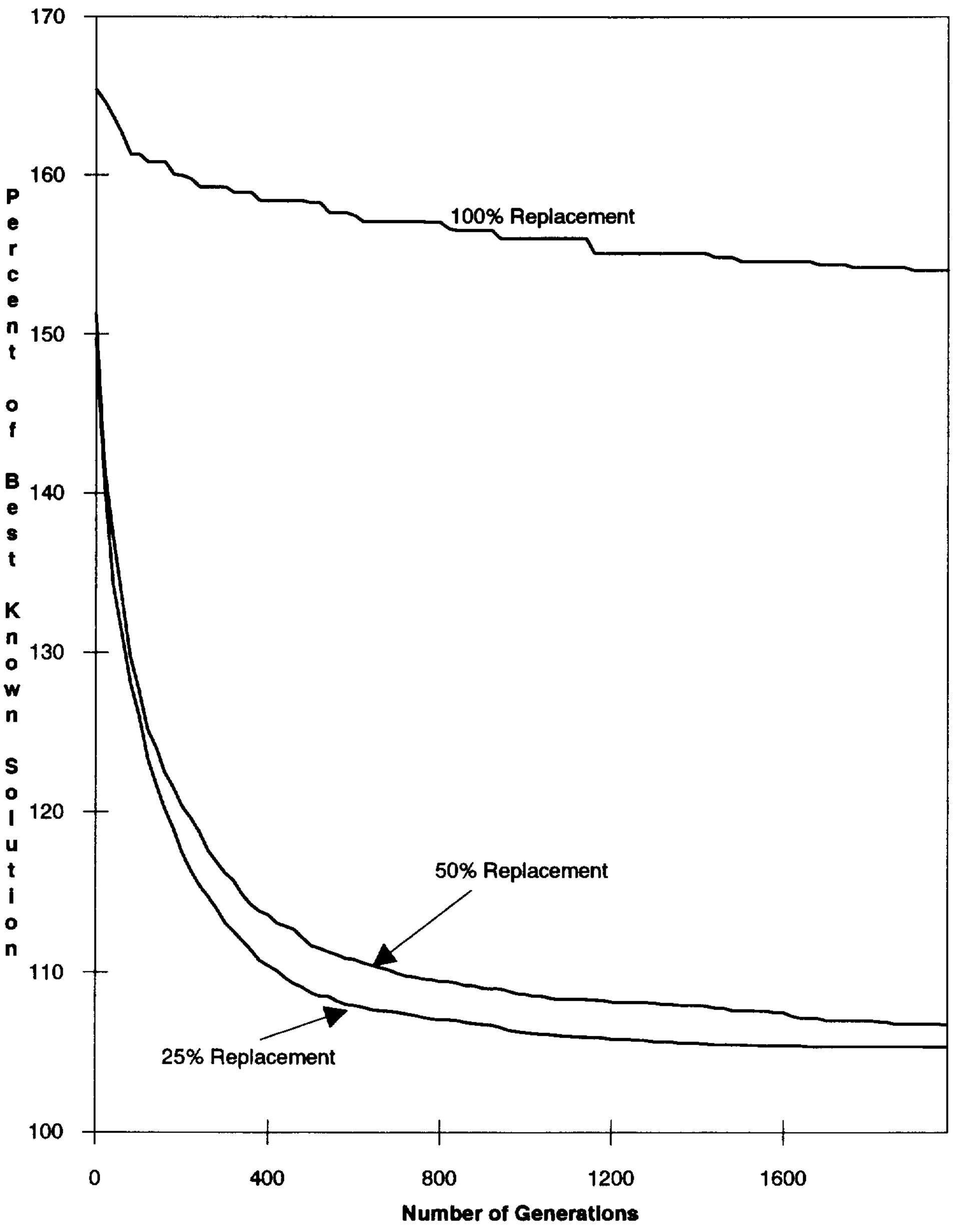

Fig. 4. Pure mutation replacement schemes for the 34 object problem.

runs with both reproduction and mutation. The continued survival of a large fraction of the population's best individuals has a pronounced effect on both the quality of solutions obtained and the speed with which those solutions are found.

Our second general conclusion has to do with the preservation of feasibility in generating offspring and mutants from existing feasible solutions. Some previous GA implementations use reproduction and mutation operators which are not guaranteed to produce feasible solutions, even when applied to feasible solutions. Since infeasible offspring are generally discarded the population remains unchanged, and the implementation incurs all the computational penalties of search, but none of the potential benefits. We believe that the combinatorial size of most problems makes it imperative that GA reproduction and mutation only produce feasible encodings. Consider for the relatively small 15-site problem, there are 15! ($\approx 1.3 \times 10^{12}$) different sequences. Even allowing for symmetric configurations, there are more than $10^{11}$ distinct feasible assignments. And yet our GA, in examining $<200{,}000$ of these solutions (or 0.0000006 of the total), often equaled the best known solution.

As a further example, consider the 16-site problem described by Cohoon *et al.* [4]. They present this as a special case of a general problem involving determining the shape and placement of computer components, subject to constraints on allowable aspect ratio. For the general problem, they consider costs arising from interconnection of components (as in QAP) and from the total area of all components. Their encoding is a representation of an arbitrary sequence of bisecting refinements of a planar partition. In the special case of QAP, this requires the algorithm not only to find the optimal assignment of components to sites in a rectangular lattice, but also to recognize that a rectangular lattice is the only feasible configuration which uses minimum area. To further complicate matters, it is not clear that the mutation and reproduction operators always preserve feasibility. As a result, for comparable computational effort (200,000 solutions generated), the Cohoon GA is strongly dominated by our GA implementation. This is not an indictment of the Cohoon method; it is merely evidence that there are good reasons to choose an encoding which is as problem-specific as is possible for any known special structure of the problem instance at hand, and that any work expended in generating solutions infeasible for this instance carries a high penalty.

Finally, it should be noted that it became obvious fairly early in the experimental testing that certain mixes of mutation and reproduction consistently outperformed other mixes for these problems. Had we merely been interested in finding the best overall solution for a given amount of computational effort, we would have dynamically adjusted our mix as we became aware of these trends. Certaintly, there was no reason to continued running 25% mutation runs except for completeness of results reported in this paper. Similarly, observation of the convergence rates of various runs would have led us to adjust our number of generations per run downward, saving CPU time with little or no cost in objective function value. We might very well have obtained better solutions by running half as many generations with twice as many seeds. We leave quantification of this effect for further research.

In summary, we have shown that our genetic approach to QAP yields solutions comparable to those of the best previously reported heuristics without extreme computational requirements. The best solutions found over a moderate number of test runs were invariably within a few percent of the best known solutions, and the average quality of the GA solutions were at least as good as those from previous heuristics for which average cases were reported. We note that there appear to be strong advantages to encodings and generation operators which are as problem specific as possible, and which preserve feasibility in all cases. From the observed robustness of the algorithm and encoding, we conjecture that extension of the GA methodology to more complex assignment and placement problems will also be effective and practical.

*Acknowledgements*—The authors would like to acknowledge the helpful comments of two anonymous reviewers.

## REFERENCES

1. M. S. Bazaraa and O. Kirca, A branch-and-bound heuristic for solving the QAP. *Naval Res. Logist. Q.* **30**, 287–304 (1983).
2. M. Beghin-Picavet and P. Hansen, Deux problèmes d'affectation non linéaires. *RAIRO Recherche Opérationelle* **16**, 263–276 (1982).
3. R. E. Bukard and T. Bonniger, A heuristic for quadratic boolean program with applications to quadratic assignment problems. *Eur. J. Opns Res.* **13**, 374–386 (1983).
4. J. P. Cohoon, S. U. Hegde, W. N. Martin and D. S. Richards, Distributed genetic algorithms for the floorplan design problem. *IEEE Trans. Computer-Aided Design* **10**, 483–492 (1991).
5. J. P. Cohoon and W. D. Paris, Genetic placement. *IEEE Trans. Computer-Aided Design* **6**, 956–964 (1987).
6. L. Davis, Job shop scheduling with genetic algorithms. *Proc. Int. Conf. Genetic Algorithms and Their Applications*, pp. 136–140 (1985).
7. L. Davis and S. Coombs, Genetic algorithms and communications link speed design: theoretical considerations. *Proc. Second Int. Conf. Genetic Algorithms*, pp. 252–256 (1987).
8. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco (1979).
9. M. Hanan and J. M. Kurtzberg, A review of the placement and quadratic assignment problems. *SIAM Rev.* **14**, 324–342 (1972).
10. S. S. Heragu and A. Kusiak, Machine layout problem in flexible manufacturing systems. *Opns Res.* **36**, 258–268 (1988).
11. F. S. Hillier and M. M. Connors, Quadratic assignment problem algorithms and the location of indivisible facilities. *Mgmt Sci.* **13**, 42–57 (1966).
12. G. G. Hitchings and M. Cottam, An efficient heuristic procedure for solving the layout design problem. *Omega* **4**, 205–214 (1976).
13. J. H. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor (1975).
14. E. S. H. Hou, R. Hong and N. Ansari, Efficient multiprocessor scheduling based on genetic algorithms. *Proc. IECON '90*, Vol 2, pp. 1239–1243 (1990).

15. E. S. H. Hou and H.-Y. Li, Task scheduling for flexible manufacturing systems based on genetic algorithms. *Proc. 1991 IEEE Int. Conf. Systems, Man, and Cybernetics*, pp. 397–402 (1991).
16. T. S. Hundal, The facility layout problem: a myopic priority heuristic. Ph.D. dissertation, University of Pittsburgh (1991).
17. B. K. Kaku, G. L. Thompson and T. E. Morton, A hybrid heuristic for the facilities layout problem. *Comput. Opns Res.* **18**, 241–253 (1991).
18. T. C. Koopmans and M. Beckmann, Assignment problems and the location of economic activities. *Econometrica* **25**, 53–76 (1957).
19. C. E. Nugent, T. E. Vollmann and J. Ruml, An experimental comparison of techniques for the assignment of facilities to locations. *Opns Res.* **16**, 150–173 (1968).
20. C. J. Picone and W. E. Wilhelm, A perturbation scheme to improve Hillier's solution to the facilities location problem. *Mgmt Sci.* **30**, 1238–1249 (1984).
21. H. T. Sieglemann and O. Frieder, The allocation of documents in multiprocessor information retrieval systems: an application of genetic algorithms. *Proc. 1991 IEEE Int. Conf. Systems, Man, and Cybernetics*, pp. 645–650 (1991).
22. L. Steinberg, The backboard wiring problem: a placement algorithm. *SIAM Rev.* **3**, 37–50 (1961).
23. M. R. Wilhelm and T. L. Ward, Solving quadratic assignment problems by 'simulated annealing'. *IIE Trans.* **19**, 107–119 (1987).

## APPENDIX

*One of Our Best Solutions to Each Test Problem*

| | | |
|---|---|---|
| 4 | 2 | D |
| 5 | 1 | 3 |

5 Object / 6 Site Rectilinear
Cost = 25.

| | | |
|---|---|---|
| 3 | 2 | 1 |
| 4 | 5 | 6 |

6 Object / 6 Site Rectilinear
Cost = 43.

| | | |
|---|---|---|
| 5 | 3 | 2 |
| D | 6 | 1 |
| D | 7 | 4 |

7 Object / 9 Site Rectilinear
Cost = 74.

| | | | |
|---|---|---|---|
| 7 | 6 | 5 | 3 |
| 8 | 4 | 1 | 2 |

8 Object / 8 Site Rectilinear
Cost = 107.

| | | | |
|---|---|---|---|
| 3 | 9 | 6 | 12 |
| 1 | 11 | 5 | 4 |
| 2 | 10 | 7 | 8 |

12 Object / 12 Site Rectilinear
Cost = 289.

| | | | | |
|---|---|---|---|---|
| 12 | 5 | 10 | 15 | 6 |
| 11 | 9 | 14 | 3 | 4 |
| 7 | 8 | 13 | 2 | 1 |

15 Object / 15 Site Rectilinear
Cost = 575.

| | | | | |
|---|---|---|---|---|
| 13 | 1 | 11 | 20 | 7 |
| 10 | 9 | 12 | 14 | 3 |
| 5 | 8 | 15 | 2 | 6 |
| 19 | 16 | 4 | 17 | 18 |

20 Object / 20 Site Rectilinear
Cost = 1299.

| | | | | | |
|---|---|---|---|---|---|
| 14 | 4 | 30 | 27 | 18 | 15 |
| 23 | 29 | 24 | 11 | 20 | 17 |
| 3 | 16 | 12 | 8 | 21 | 1 |
| 22 | 10 | 13 | 9 | 7 | 19 |
| 5 | 2 | 28 | 25 | 6 | 26 |

30 Site / 30 Object Rectilinear
Cost = 3092.

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 |

16 Object / 16 Site Rectilinear
Cost = 48.

| | | | | |
|---|---|---|---|---|
| 16 | 25 | 24 | 23 | 22 |
| 15 | 17 | 18 | 19 | 21 |
| 6 | 8 | 13 | 14 | 20 |
| 5 | 7 | 12 | 9 | 11 |
| 4 | 3 | 2 | 1 | 10 |

25 Object / 25 Site Euclidean
Cost = 1624.6.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 33 | 34 | 31 | 30 | 29 | 15 | 3 | 16 | D |
| 26 | 32 | 28 | 14 | 12 | 13 | 5 | 18 | 17 |
| 25 | 19 | 23 | 20 | 11 | 7 | 4 | 8 | 2 |
| 24 | 21 | 22 | 27 | 6 | 1 | 10 | 9 | D |

34 Object / 36 Site Euclidean
Cost = 4271.5.