Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE @ DIRECT®

Knowledge-Based Systems 16 (2003) 261–268

Knowledge-Based SYSTEMS

www.elsevier.com/locate/knosys

Genetic algorithm hybridized with ruin and recreate procedure: application to the quadratic assignment problem

Alfonsas Misevicius*

Department of Practical Informatics, Kaunas University of Technology, Studentu St. 50-416D, Kaunas LT-3031, Lithuania

Abstract

Genetic algorithms (GAs) are among the widely used in various areas of computer science, including optimization problems. In this paper, we propose a GA hybridized with so-called ruin and recreate (R and R) procedure. We have applied this new hybrid strategy to the well-known combinatorial optimization problem, the quadratic assignment problem (QAP). The results obtained from the experiments on different QAP instances show that the proposed algorithm belongs to the best heuristics for the QAP. The power of this algorithm is also demonstrated by the fact that the new best known solutions were found for several QAP instances.

Keywords: Genetic algorithms; Optimization; Quadratic assignment problem

1. Introduction

Genetic algorithms (GAs) are based on the biological process of natural selection. The original concepts of GAs were developed as far back as 1975 by Holland [1]. Many simulations have demonstrated the efficiency of GAs on different optimization problems, among them, bin-packing [2], generalized assignment problem [3], graph partitioning [4], job-shop scheduling problem [5], set covering problem [6], traveling salesman problem [7], vehicle routing [8].

The very basic definitions related to GAs and optimization problems are as follows. Let S be a set of solutions of a combinatorial optimization problem with objective (fitness) function $f : S \to R^1$. Furthermore, let P be a subset of S. It is referred to as a population, and it is composed of individuals (solutions), $s_1, s_2, \ldots$ Without loss of generality, we assume that the objective function seeks a global minimum; hence, individual s' is preferred to individual s'' if $f(s') < f(s'')$. GAs can then be described as follows. A fraction of P is selected to be parents by use of the selection function, $\phi : 2^S \to S \times S$. New solutions are created by combining pairs of parents; this recombination operator, $\psi : S \times S$, is known as a crossover. Afterward, a culling operator, $\varphi : 2^S \to 2^S$, is applied to the previous generation and the offsprings to determine which individuals survive to form the next generation. In addition, the individuals of the population undergo a mutation, $\zeta : S \to S$, to prevent a premature loss of diversity within the population.

There exist many variants in the choice of how select, cross (recombine), mutate, and cull the individuals of the population. For example, one can maintain one large population, or several smaller, parallel sub-populations. One can choose to replace all the individuals of the current population with offsprings of every generation, or one can replace only the worst individual by the best offspring. When designing crossover, mutation procedures, many more variations and modifications are available. For the details, the reader is addressed to Refs. [9,10,11].

This paper is organized as follows. In Section 2, a hybrid genetic approach is outlined. Section 3 describes the application of the proposed hybrid algorithm to the quadratic assignment problem (QAP). The results of the computational experiments on the various QAP instances are presented in Section 4. Section 5 completes the paper with concluding remarks.

2. Genetic algorithm hybridized with ruin and recreate procedure

The standard GAs rely on the concept of biological evolution. State-of-the-art GAs, often referred to as hybrid GAs, are based rather on cultural-idea-evolution. While in

* Tel.: +370-7-30-03-72; fax: +370-7-30-03-52.
E-mail address: misevi@soften.ktu.lt (A. Misevicius).

0950-7051/03/$ - see front matter © 2003 Elsevier Science B.V. All rights reserved.
doi:10.1016/S0950-7051(03)00027-3

nature genes are typically not changed during an individual's lifetime, ideas are usually modified before passing to the next generation. In contrast to classical GAs, hybrid GAs incorporate a mechanism for the improvement of the individuals at every generation. An improvement procedure is to be applied to each offspring and/or mutated individual. All the solutions of the initial population are also improved before starting the iterative process of GA. So, instead of (random) solutions, hybrid GA operates with the improved-elite-solutions. This leads to a more effective algorithm when comparing it with a pure GA.

In principle, any local search concept based algorithm can be applied at the improvement phase of the hybrid GA. In the simplest case, a greedy (or steepest) descent (also known as 'hill climbing') algorithm can be used. Yet, it is possible to apply more sophisticated algorithms, like simulated annealing or tabu search. In this paper, we propose to hybridize the GA with a procedure that is based on so-called ruin and recreate (R and R) principle [12].

The heart of the R and R approach, which bears resemblance to a variable neighbourhood search [13], is to obtain better optimization results by a destruction (reconstruction) of an existing solution and a following rebuilding (improvement) procedure. By applying this type of treatment frequently, one seeks for high quality solutions. So, R and R can be thought of as an iterative process of reconstructions and improvements applied to solutions. The advantage of the R and R approach over the well-known random multistart method is that, instead of generating new solutions from scratch, a better idea is to reconstruct (a part of) the current solution: continuing search from this reconstructed solution may allow to escape from a local optimum and to find better solutions.

One of the favourable features of the proposed hybrid strategy is that the large population of solutions is not necessary: small size of the population is compensated by high performance of the ruin and recreate procedure. Another distinguishing feature is related to the mutation. As the R and R procedure has a stochastic behaviour, there is no need in mutation operator unless the population undergoes some kind of reconstruction. It takes place when a premature convergence of the algorithm occurs. As

```
procedure hybrid_genetic_algorithm
  generate (or construct) initial population $P$
  foreach $s \in P$ do ruin_and_recreate($s$,"recreate first")
  repeat // main cycle //
    for $c$:=1 to $C_{offspr}$ do begin // cycle of offspring generation and improvement //
      select $s', s'' \in P$
      $s'''$ := crossover($s', s''$)
      ruin_and_recreate($s'''$,"recreate first")
      $P := P \cup \{ s''' \}$
    end // for //
    update $P$
    if entropy($P$)<threshold then foreach $s \in P \backslash \{s_{best}\}$ do ruin_and_recreate($s$,"ruin first")
  until termination criterion is satisfied
end // hybrid_genetic_algorithm //
```

Fig. 1. Template of the hybrid genetic algorithm. Note. The function 'entropy' is described in Section 3.2.5.

```
procedure ruin_and_recreate($s$,option)
  if option="ruin first" then $s$ := ruin($s$)
  $s^*$ := recreate($s$)
  $s := s^*$
  for $q$:=1 to $Q_{RR}$ do begin // cycle consisting of $Q_{RR}$ iterations //
    $s^{\sim}$ := ruin($s$)          // ruin (reconstruct) the current solution (individual) //
    $s^*$ := recreate($s^{\sim}$)    // try to improve the ruined solution //
    if $s^*$ is better than $s$ then $s := s^*$
  end // for //
end // ruin_and_recreate //
```

Fig. 2. Template of the ruin and recreate procedure.

a convergence criterion, a measure (threshold) of entropy of the population is used. The templates of both the resulting hybrid GA and the R and R procedure are presented in Figs. 1 and 2. More thoroughly, we discuss the components of the hybrid GA in Section 3 in the context of the quadratic assignment problem.

3. Application to the quadratic assignment problem

3.1. The quadratic assignment problem

The quadratic assignment problem is formulated as follows. Let two matrices $\mathbf{A} = (a_{ij})_{n\times n}$ and $\mathbf{B} = (b_{kl})_{n\times n}$ and the set $\prod$ of permutations of the integers from 1 to n be given. Find a permutation $\pi = (\pi(1), \pi(2), \dots, \pi(n)) \in \prod$ that minimizes

$$z(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{\pi(i)\pi(j)}.$$

The context in which Koopmans and Beckmann [14] first formulated this problem was the facility location problem. In this problem, one is concerned with locating n units on n sites with some physical products flowing between the facilities, and with distances between the sites. The element a_{ij} is the 'flow' from unit i to unit j, and the element b_{kl} represents the distance between the sites k and l. The permutation $\pi = (\pi(1), \pi(2), \dots, \pi(n))$ can be interpreted as an assignment of units to sites ($\pi(i)$ denotes the site what unit i is assigned to). Solving the QAP means searching for an assignment π that minimizes the 'transportation cost' z.

An important area of the application of the quadratic assignment problem is computer-aided design (CAD), more precisely, the placement of electronic components into locations (positions) on a board (chip) [15]. Other applications of the QAP are: campus planning [16], hospital layout [17], image processing (formation of grey densities) [18], typewriter keyboard design [19], etc. [20].

It has been proved that the QAP is NP-hard [21]. Problems of size, say $n > 30$, are not, to this date, practically solvable in terms of obtaining exact solutions. Therefore, heuristic approaches have to be used for solving medium-and large-scale QAPs: ant algorithms [22], simulated annealing

[23], tabu search [24]. Starting from 1994, several authors applied GAs to the QAP, first of all, [25–28].

3.2. A hybridized genetic algorithm for the quadratic assignment problem

All we need to do by creating our GA (we will call it GAHRR—GA Hybridized with Ruin and Recreate procedure) is to design the genetic operators and ruin and recreate procedure as a local improvement procedure. The details of these components (in the context of the quadratic assignment problem) are described below.

3.2.1. Initial population generation

The initial population P ($P \subset \prod$) is obtained by generating random permutations that are passed through the improvement (ruin and recreate) procedure—the same one that is used at further phase, i.e. within the genetic operators. The size of the population, PS ($\mathrm{PS} = |P|$), depends on the size of problem (n), and is relatively small (we tried the sizes in the range $[\lfloor\sqrt{n}\rfloor, 2\lfloor\sqrt{n}\rfloor]$.

3.2.2. Selection rule

For the parents selection, we applied a rank based selection rule [28]. In this case, the position, u, of the parent (solution) actually selected within the sorted population is determined according to the formula $u = \lfloor v^\rho \rfloor$, where v is a uniform real random number from the interval $[1, \mathrm{PS}^{1/\rho}]$, where PS is the population size, and ρ is a real number in the interval [1,2] referred to as a selection factor (SF). It is obvious that the larger the value of SF, the larger probability of selecting better parents.

3.2.3. Crossover operator

The crossover operator produces an offspring (child) that shares some characteristics from both of his parents. The permutation structure is to be preserved. Our crossover is based on the one due to Tate and Smith [28]. Its very formal description is as follows

```
TSX($\pi_1$, $\pi_2$)//Tate and Smith's crossover//
//input: $\pi_1$, $\pi_2$—parents; output: o—offspring ($\pi_1$, $\pi_2$, o $\in \prod$)//
$\forall i \in \{1, 2, \ldots, n\}$ :
    $o(i) :=$ if(randint(1,2) = 1, if($\pi_1(i)$ not assigned, $\pi_1(i)$, 0), if($\pi_2(i)$ not assigned, $\pi_2(i)$, 0)),
    once $\pi_1(i)$ (or $\pi_2(i)$) is assigned it is no longer considered in future assignments;
$\forall i \in \{1, 2, \ldots, n\}$ :
    if $o(i) = 0$ then $o(i) := j | j \in \{k | k$ is not assigned$\}$
end//TSX//
```

Notes (1) The function if(x, y_1, y_2) returns y_1 if $x =$ TRUE, otherwise it returns y_2. (2) The function randint (x, y) returns integer random, uniformly distributed number between x and y.

We have slightly enhanced the TSX crossover as described below

```
OTSX($\pi_1$, $\pi_2$)//optimized Tate and Smith's crossover//
//input: $\pi_1$, $\pi_2$—parents; output: o—offspring ($\pi_1$, $\pi_2$, o $\in \prod$)//
$o := \arg\min_{i=1,\ldots,M} z(o_i)$, where $o_i$ is returned by $i$ th call of TSX crossover
end//OTSX//
```

So, only the best child among M children—elite child—is passed through the subsequent improvement procedure. The value of M should not be very large ($0.1n \le M \le n$, for example, $M = 0.3n$).

The number of crossovers, i.e. the offsprings created per one generation is controlled by a corresponding parameter, C_{offspr}. Usually, C_{offspr} is between 1 and 4 when dealing with small populations.

3.2.4. Improvement procedure: ruin and recreate procedure

Ruin and recreate procedure contains three main components: a solution recreation procedure (local search procedure), a ruin procedure (solution mutation (or reconstruction) procedure), and a candidate acceptance criterion.

Recreation procedure. In our algorithm, we use the tabu search methodology based algorithm as a recreation procedure, more precisely, a modified version of the robust tabu search (RTS) algorithm due to Taillard [24].

Let $N: \prod \rightarrow 2^{\Pi}$ be a neighbourhood function that defines for each $\pi \in \prod$ a set $N(\pi) \subseteq \prod$—a set of neighbouring solutions of π. A λ-exchange neighbourhood function, $N_\lambda(\pi)$, is defined as follows $N_\lambda(\pi) = \{\pi' | \pi' \in \prod, d(\pi, \pi') \le \lambda\}$, where $d(\pi, \pi')$ is the 'distance' between solutions π and π': $d(\pi, \pi') = \sum_{i=1}^{n} \mathrm{sgn}|\pi(i) - \pi'(i)|$. For the QAP, the commonly used case is $\lambda = 2$, i.e. the 2-exchange function N_2. In this case, a transformation from the current solution π to the neighbouring one π' can formally be defined by using a special operator–2-way perturbation, p_{ij} $(i, j = 1, 2, \ldots, n) : \prod \rightarrow \prod$, which exchanges i th and j th elements in the current permutation. (Notation $\pi' = \pi \oplus p_{ij}$ means that π' is obtained from π by applying perturbation p_{ij}.) The framework of the local search algorithm can then be described as follows. Initialize tabu list, $\mathbf{T} = (t_{ij})_{n \times n}$, and start from an initial solution π. Continue the following process until a predetermined number of trials is executed: (a) find a neighbour π'' of the current solution π in such a way that $\pi'' = \arg\min_{\pi' \in N_2'(\pi)} z(\pi')$, where $N_2'(\pi) = \{\pi' | \pi' \in N_2(\pi), (\pi' = \pi \oplus p_{ij}$ and p_{ij} is not tabu or $(z(\pi') < z(\pi'''))\}$ (π''' is the best so far solution); (b) replace the current solution π by the neighbour π'', and use as a starting point for the next trial; (c) update the tabu list $\mathbf{T}$. The detailed template of the tabu search based recreation procedure for the QAP is presented in Appendix A. Note: the parameter α controls how many recreation trials one has to perform.

Ruin procedure. A ruin (or mutation) of the current solution is achieved by performing a 'move' in the neighbourhood N_λ, where $\lambda > 2$. This can be modelled by generating $\mu = \lfloor(\lambda + 1)/2\rfloor$ sequential elementary perturbations, like p_{ij}. Here, the parameter μ is referred to as a ruin rate ('strength'). We can add more robustness to the R and R procedure if we let vary the parameter μ in some interval, say $[\mu_{\min}, \mu_{\max}] \subseteq [2, n]$, during the execution of the algorithm. In our implementation, μ is varied sequentially within the interval $[\mu_{\min}, \mu_{\max}]$ starting from $\mu_{\min}$: once maximum value $\mu_{\max}$ has been reached (or a better locally optimum solution has been found) the value of μ is dropped to the minimum value $\mu_{\min}$, and so on. The detailed template of the ruin procedure is presented in Appendix B. Note that, for the convenience, theparameters $\beta_{\min}$, $\beta_{\max} \in (0, 1)$ are used instead of $\mu_{\min}$, $\mu_{\max}$, respectively, ($\mu_{\min} = \max(2, \lfloor\beta_{\min}\cdot n\rfloor)$, $\mu_{\max} = \max(2, \lfloor\beta_{\max}\cdot n\rfloor)$).

Candidate acceptance criterion. In our algorithm, we accept only the best locally optimal solution as a candidate for the ruin (mutation). The candidate solution at qth iteration is defined according to the following formula $\pi^{(q)} = \mathrm{if}(z(\pi^\bullet) < z(\pi^{(q-1)}), \pi^\bullet, \pi^{(q-1)})$, where $\pi(^q)$ is the candidate solution at the current iteration, $\pi^{(q-1)}$ is the solution before the last recreation execution (the best so far solution), and $\pi^\bullet$ is the solution after the last recreation execution.

The number of iterations, Q_{RR}, plays a role of the termination criterion for the R and R procedure. The number of iterations of the R and R procedure used at the initial population generation (initialization) phase is equal to $R\cdot Q_{\mathrm{RR}}$, where $R(R \geq 1)$ is a factor (aspect ratio).

3.2.5. Updating the population

As a standard, updating (culling) of the population takes place each time before going to the next generation. After $k(k = C_{\mathrm{offspr}})$ offsprings have been added to the population, its members are sorted (according to the increasing values of the objective function); k solutions with the greatest objective function value are then removed away from the population to keep the size of the population constant. A different kind of the update is performed when the premature convergence of the algorithm is observed. We identify the convergence by means of an entropy measure. The entropy of the population is defined in the following way [25]

$$E = \sum_{i=1}^{n} \sum_{j=1}^{n} \mathrm{e}_{ij}/n \log n,$$

where

$$\mathrm{e}_{ij} = \begin{cases} 0, m_{ij} = 0 \\ -\dfrac{m_{ij}}{m} \log \dfrac{m_{ij}}{m}, m_{ij} > 0 \end{cases},$$

m is the size of the population, and m_{ij} represents the number of times that the unit i is assigned to the site j in the current population.

This normalized entropy takes values between 0 and 1. So, if E is close enough to zero ($E \leq 0.01$), we state that the algorithm has converged—further improvements of the individuals are unlikely. In this case, we update the population in such a way that all the individuals of the population but the best undergo a deep reconstruction (mutation). The R and R procedure with the ruin being performed at the start serves as a reconstruction procedure. In addition, the ruin rate as well as the number of iterations of R and R are slightly increased. After the reconstruction, the algorithm proceeds with the new population in an ordinary way.

The process is continued until the stopping criterion is satisfied, i.e. the algorithm has performed a total number of N_{gen} generations.

4. Computational experiments and results

We have carried out a number of computational experiments in order to test the performance of the algorithm GAHRR. A wide range of the QAP instances taken from the quadratic assignment problem library QAPLIB [29] were used. The types of the QAP instances that we use are as follows:

(a) randomly generated instances (these instances are randomly generated according to a uniform distribution; in QAPLIB, they are denoted by tai20a, tai25a, tai30a, tai35a, tai40a, tai50a, tai60a, tai80a, tai100a (short notation tai*a));
(b) real-life like instances (they are generated in such a way that the entries of the data matrices resemble a distribution from real-life problems; the instances are as follows: tai20b, tai25b, tai30b, tai35b, tai40b, tai50b, tai60b, tai80b, tai100b (tai*b)).

For the comparison, we used the RTS algorithm due to Taillard [24], and the genetic hybrid algorithm (GHA) due to Fleurent and Ferland [25]. The first one belongs to the most powerful algorithms for the problems of type (a), whereas the second one is among the best algorithms for the problems of type (b).

As a performance measure, the average deviation from the best known solution is chosen. The average deviation, θ_{avg}, is defined according to the formula $\theta_{\mathrm{avg}} = 100(z_{\mathrm{avg}} - z_b)/z_b[\%]$, where z_{avg} is the average objective function value over $W(W = 1, 2, \ldots)$ restarts (i.e. single applications of the algorithm to a problem instance), and z_b is the best known value (BKV) of the objective function. (BKVs are from Ref. [29]).

The values of the parameters of the algorithm GAHRR for the particular problem types are presented below

Problem types	Parameters							
	PS	N_{gen}	C_{offspr}	SF	Q_{RR}	R	α	$(\beta_{\min}, \beta_{\max})$
(a)	$\lfloor\sqrt{n}\rfloor$	30	1	1.1	4	3	10	$\begin{cases}(0.2,0.3), n \geq 50\\(0.25,0.35), 30 \leq n < 50\\(0.3,0.4), n < 30\end{cases}$
(b)	$2\cdot\lfloor\sqrt{n}\rfloor$	80	4	1.7	8	3	0.01	(0.45, 0.55)

The parameters of the algorithm GHA are similar to those of GAHRR: the population size, PS' (it is equal to $\min(100, 2n)$); the number of generations, N'_{gen}; the number of offsprings per generation, $C'_{\text{offspr}}(C'_{\text{offspr}} = 2)$; the selection factor, $SF'(SF' = 2.0)$; the number of iterations of the improvement procedure, i.e. RTS procedure, Q'_{RTS} (it is chosen to be equal to $4n$). The same number of iterations is both in initialization phase and in subsequent evolution phase.

The only control parameter of RTS is the number of iterations, Q_{RTS}. The value of Q_{RTS} as well as the value of N'_{gen} were chosen in such a way that all the algorithms use approximately the same computation (CPU) time. Some differences of run time are due to non-deterministic behaviour of the tabu search based recreation procedure used in GAHRR. In most cases, they are insignificant and cannot influence the results obtained.

The results of the comparison, i.e. the average deviations from BKV, as well as the approximated CPU times per restart (in seconds) are presented in Tables 1 and 2. The deviations are averaged over 10 restarts. CPU times are given for the x86 Family 6 processor. The best values are printed in bold face (in addition, in parenthesis, we give the number of times that BKV is found (if $\theta_{\text{avg}} > 0$)).

It turns out that the efficiency of the algorithms depends on the type of the problems (instances) being solved. For the randomly generated instances (instance family tai*a), the results are not as good as for real-life like instances (instance family tai*b). Nevertheless, the results from Tables 1 and 2 show promising efficiency of the proposed hybrid GA. In most cases, the algorithm GAHRR appears to be superior to other two efficient algorithms. For a number of instances, for example, tai20a, tai25a, tai30a, tai80b, GAHRR produces much more better results than other algorithms (the record difference in results was achieved for the instance tai80b (Table 2)).

The performance of the algorithm GAHRR can be improved either by a more accurate tuning of the control parameters (we call this direction 'the parameter improvement based direction'), or modifying some components of the algorithm ('the algorithm's idea improvement based direction', some directions of the enhancement of the algorithm's idea are outlined in Section 5). The results obtained by using different values of the parameters PS, C_{offspr}, SF and Q_{RR} are presented in Table 3. The values of the parameters are as follows

$$PS = 2\cdot\lfloor\sqrt{n}\rfloor, C_{\text{offspr}} = \lfloor 0.25\cdot PS\rfloor, SF = 1.7,$$
$$Q_{\text{RR}} = \lfloor 0.1n\rfloor \text{ (column a);}$$
$$PS = 3\cdot\lfloor\sqrt{n}\rfloor, C_{\text{offspr}} = \lfloor 0.25\cdot PS\rfloor, SF = 1.7,$$
$$Q_{\text{RR}} = \lfloor 0.1n\rfloor \text{ (column b);}$$
$$PS = 2\cdot\lfloor\sqrt{n}\rfloor, C_{\text{offspr}} = \lfloor 0.5\cdot PS\rfloor, SF = 1.7,$$
$$Q_{\text{RR}} = \lfloor 0.1n\rfloor \text{ (column c);}$$
$$PS = 2\cdot\lfloor\sqrt{n}\rfloor, C_{\text{offspr}} = \lfloor 0.25\cdot PS\rfloor, SF = 2.0,$$
$$Q_{\text{RR}} = \lfloor 0.1n\rfloor \quad \text{(column d);}$$
$$PS = 2\cdot\lfloor\sqrt{n}\rfloor, C_{\text{offspr}} = \lfloor 0.25\cdot PS\rfloor, SF = 1.7,$$
$$Q_{\text{RR}} = \lfloor 0.15n\rfloor \quad \text{(column e).}$$

Table 1
Comparison of the algorithms on randomly generated instances

Instance name	n	BKV	RTS		GHA		GAHRR	
			θ_{avg}	t	θ_{avg}	t	θ_{avg}	t
tai20a	20	703482	$0.061_{(8)}$	3.5	$0.411_{(2)}$	3.3	**0**	3.2
tai25a	25	1167256	$0.052_{(8)}$	8.3	$0.382_{(2)}$	7.7	**0**	7.4
tai30a	30	1818146	$0.044_{(6)}$	14.5	$0.362_{(4)}$	13.4	$\mathbf{0.002}_{(9)}$	12.5
tai35a	35	2422002	$0.327_{(0)}$	23	$0.643_{(0)}$	21	$\mathbf{0.163}_{(5)}$	19
tai40a	40	3139370	$0.556_{(0)}$	25	$0.618_{(0)}$	33	$\mathbf{0.451}_{(0)}$	25
tai50a	50	4941410	$0.962_{(0)}$	52	$0.871_{(0)}$	55	$\mathbf{0.668}_{(1)}$	49
tai60a	60	7208572	$1.122_{(0)}$	83	$1.047_{(0)}$	86	$\mathbf{0.910}_{(0)}$	82
tai80a	80	13557864	$0.825_{(0)}$	200	$0.917_{(0)}$	190	$\mathbf{0.739}_{(0)}$	190
tai100a	100	21125314	$0.731_{(0)}$	430	$0.967_{(0)}$	440	$\mathbf{0.629}_{(0)}$	450

Table 2
Comparison of the algorithms on real-life like instances

Instance name	n	BKV	RTS		GHA		GAHRR	
			θ_{avg}	t	θ_{avg}	t	θ_{avg}	t
tai20b	20	122455319	**0**	3.4	$0.045_{(9)}$	4.1	**0**	3.1
tai25b	25	344355646	**0**	5.9	**0**	7.8	**0**	5.6
tai30b	30	637117113	$0.000_{(8)}$	10	**0**	13.5	**0**	9.7
tai35b	35	283315445	**0**	17	$0.094_{(6)}$	22	**0**	15
tai40b	40	637250948	**0**	25	**0**	27	**0**	27
tai50b	50	458821517	$0.114_{(4)}$	47	$0.033_{(9)}$	55	**0**	49
tai60b	60	608215054	$0.125_{(2)}$	81	$0.014_{(6)}$	87	**0**	82
tai80b	80	818415043	$0.394_{(0)}$	185	$0.353_{(2)}$	190	$\mathbf{0.001}_{(5)}$	195
tai100b	100	1185996137	$0.279_{(0)}$	390	$0.045_{(3)}$	370	$\mathbf{0.015}_{(3)}$	410

R, α, $\beta_{\min}$, $\beta_{\max}$ remain unchanged

$$R = \begin{cases} 2, n \le 30 \\ 3, 30 < n \le 80, \\ 4, n = 100 \end{cases} \qquad \alpha = \begin{cases} 0.1, n \le 25 \\ 0.2, 25 < n \le 40 \\ 0.3, 40 < n \le 80 \\ 0.5, n = 100 \end{cases},$$

$$\beta_{\min} = 0.45, \ \beta_{\max} = 0.55.$$

Table 3
Computational results of GAHRR

Instance name	t_{avg}				
	(a)	(b)	(c)	(d)	(e)
tai20b	**0.05**	0.06	**0.05**	0.06	0.06
tai25b	0.24	0.22	**0.2**	0.26	0.28
tai30b	0.6	0.6	0.55	**0.45**	0.65
tai35b	2.2	2.4	**1.7**	6.6	2.3
tai40b	1.5	2.0	**1.3**	1.4	1.8
tai50b	6.6	8.9	**6.5**	7.0	13.0
tai60b	25.0	21.0	22.0	**20.0**	28.0
tai80b	148.0	176.0	**130.0**	212.0	310.0
tai100b	**690.0**	990.0	800.0	>3600	1800.0

Note t_{avg} denotes the average CPU time (in seconds) needed to find the best known solution under condition that all the 10 restarts of GAHRR succeeded in finding the best known solution.

Table 4
New best known solutions for grey density problems

Instance name	n	Previous best known value[a]	New best known value
grey_16_16_26	256	2426606	2426298
grey_16_16_45	256	8677670	8674910
grey_16_16_46	256	9134172	9129192
grey_16_16_77	256	29928042	29895152
grey_16_16_83	256	35447648	35444806
grey_16_16_84	256	36409340	36397376
grey_16_16_86	256	38393344	38376438
grey_16_16_90	256	42621464	42608826

[a] comes from Ref. [30].

Increasing the population size, as expected, does not improve the results, i.e. the average time needed to search for the best known solution. Also, the variation of the parameter *SF* does not influence the quality of the results considerably, except the instance tai100b. Somewhat unexpected results were obtained by modifying the number of iterations Q_{RR}—as Q_{RR} increases, the average search time increases as well. One of the reasons for this might be a 'stagnation phenomenon' of the ruin and recreation process when the number of iterations is increased. A better idea, therefore, is to increase the number of offsprings generated per generation. This leads to very promising results as shown in Table 3 (column c). We see that, for the small and medium instances, GAHRR finds the best known solutions extremely quickly. For the large instances, however, 'stagnations' can be observed. Nevertheless, GAHRR was successful in finding a number of new best solutions for the QAP instances corresponding to the elaboration of grey frames. Instances of this type are described in Ref. [18,30] under the name grey_n_1_n_2_m, where m is the density of the grey ($0 \le m \le n = n_1 \cdot n_2$), and $n_1 \cdot n_2$ is the size of a frame. New solutions for the instance family grey_16_16_$*$ are presented in Table 4.

5. Concluding remarks

The method that has been widely applied to many fields of computer science, among them optimization problems, is the genetic approach. In this paper, we introduce a new hybrid genetic strategy for a difficult optimization problem, the quadratic assignment problem.

The central idea is that the use of the GA succeeds in search only if it disposes of an efficient and robust local search procedure. So, we propose to hybridize the GA with a very promising procedure—the ruin and recreate procedure. The most important features of this approach are as follows: (a) a small population is enough to seek for high quality solutions, (b) the standard mutation operator is substituted by the ruin and recreate procedure, (c) mutations take place only if the entropy of the population approaches zero. All these features coupled with additional modifications of the genetic operators resulted in an algorithm (GAHRR) that allowed achieving very good results with small amount of the computation time. The results from the comparison with the RTS algorithm and other efficient GA show that the new hybrid algorithm produces better results than other two algorithms, which are among the most powerful heuristics for the QAP. The outstanding performance of GAHRR is also corroborated by the fact that the new best known solutions were found for the largest available QAPs (so-called problems of grey densities).

There are several possible directions of the modification of the proposed hybrid algorithm: (a) improving the performance of the existing components of the algorithm; (b) trying to use two or more sub-populations (instead of the single population) and allowing some sort of migrations of the individuals between sub-populations; (c) implementing other crossover operators, like a distance preserving or cycle crossover; (d) introducing additional components, for example, so-called ‘do not look bit’ mechanism.

It might also be worthy applying the GA presented above for other combinatorial optimization problems, like the traveling salesman problem or the graph-partitioning problem.

Acknowledgements

Author is grateful to Prof. E. Taillard for providing the code of the RTS algorithm. He also would like to thank Prof. C. Fleurent for the paper that helped coding the hybrid GA.

```
function recreate($\pi$,$n$,$\alpha$)  // recreation procedure (based on tabu search) for the QAP //
  // $\pi$ – the current permutation, $n$ – problem size, $\alpha$ – $n$-factor ($\alpha$>0) //
  $\tau$:=max(1,$\lfloor\alpha\cdot n\rfloor$)          // $\tau$ – the number of iterations //
  $h_{min}$:=0.4$n$, $h_{max}$:=0.6$n$  // lower and higher tabu sizes //
  $\pi^{\bullet}$:=$\pi$                      // $\pi^{\bullet}$ denotes the best so far permutation //
  for $i$:=1 to $n$–1 do for $j$:=$i$+1 to $n$ do store the objective function difference $\delta_{ij}=\Delta z(\pi,i,j)$
  $T$:=0
  $h$:=randint($h_{min}$,$h_{max}$)
  $i$:=1, $j$:=1, $k$:=1, $K$:=$n(n-1)/2$  // $K$ – size of the neighbourhood $N_2$ //
  improved:=FALSE
  while ($k\le\tau$) or improved do begin  // main cycle of the recreation //
    $\delta_{min}$:=$\infty$  // $\delta_{min}$ – minimum difference in the objective function values //
    for $l$:=1 to $K$ do begin
      $i$ := if($j<n$, $i$, if($i<n-1$, $i+1$, 1)) ,  $j$ := if($j<n$, $j+1$, $i+1$)
      tabu := if($t_{ij}\ge k$,TRUE,FALSE), aspired := if($z(\pi)+\delta_{ij}<z(\pi^{\bullet})$,TRUE,FALSE)
      if (($\delta_{ij}<\delta_{min}$) and not tabu) or aspired then begin $\delta_{min}$:=$\delta_{ij}$, $u$:=$i$, $v$:=$j$ end  // if //
    end  // for $l$ //
    if $\delta_{min}<\infty$ then begin
      $\pi$ := $\pi\oplus p_{uv}$  // replace the current permutation by the new one //
      if $z(\pi)<z(\pi^{\bullet})$ then $\pi^{\bullet}$:=$\pi$
      improved := if($\delta_{uv}$<0,TRUE,FALSE)
      for $l$:=1 to $n$–1 do for $m$:=$l$+1 to $n$ do update the difference $\delta_{lm}$
      $t_{uv}$:=$k+h$
      if $k$ mod $2h_{max}=0$ then $h$:=randint($h_{min}$,$h_{max}$)
    end  // if //
  end    // while //
  return $\pi^{\bullet}$
end  // recreate //
```

Fig. A.1. Template of the recreation procedure for the QAP.

```
function ruin($\pi$,$n$,$\mu$)  // ruin (mutation) procedure for the QAP //
  // $\pi$ – the current permutation, $n$ – problem size, $\mu$ – ruin rate ($2\le\mu\le n$) //
  $\pi^{\bullet}$:=$\pi$, new_local_optimum:=FALSE
  for $k$:=1 to $\mu$ do begin  // main cycle of the ruin procedure //
    $i$:=randint(1,$n$), $j$:=randint(1,$n$–1)
    if $i\le j$ then $j$:=$j$+1
    $\pi$ := $\pi\oplus p_{ij}$  // replace the current permutation by the new one //
    if $z(\pi)<z(\pi^{\bullet})$ then begin
      $\pi^{\bullet}$:=$\pi$, new_local_optimum:=TRUE
    end  // if //
  end    // for $k$ //
  if new_local_optimum then $\pi$:=$\pi^{\bullet}$
  return $\pi$
end  // ruin //
```

Fig. B.1. Template of the ruin procedure for the QAP.

Appendix A

Fig. A1.

Appendix B

Fig. B1.

References

[1] J.H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, MI, 1975.

[2] E. Falkenauer, A, Delchambre, A genetic algorithm for bin packing and line balancing, Proceedings of the IEEE International Conference on Robotics and Automation, (RA92), Nice, France, 1992.

[3] P.C. Chu, J.E. Beasley, A genetic algorithm for the generalized assignment problem, Comput. Oper. Res. 24 (1997) 17–23.

[4] T.N. Bui, B.R. Moon, Genetic algorithm and graph partitioning, IEEE Trans. Comput. 45 (1996) 841–855.

[5] T. Yamada, R. Nakano, A genetic algorithm applicable to large-scale job-shop problems, in: R. Männer, B. Manderick (Eds.), Parallel Problem Solving from Nature 2, North-Holland, Amsterdam, 1992, pp. 281–290.

[6] J.E. Beasley, P.C. Chu, A genetic algorithm for the set covering problem, Eur. J. Oper. Res. 94 (1996) 392–404.

[7] P. Merz, B. Freisleben, Genetic local search for the TSP: new results, Proceedings of the 1997 IEEE International Conference on Evolutionary Computation, Indianapolis, USA, 1997.

[8] J. Blanton Jr., R.L. Wainwright, Multiple vehicle routing with time and capacity constraints using genetic algorithms, in: S. Forrest (Ed.), Proceedings of Fifth International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, 1993.

[9] L. Davis, Handbook of Genetic Algorithms, Van Nostrand, New York, 1991.

[10] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Reading, MA, 1989.

[11] H. Mühlenbein, Genetic algorithms, in: E. Aarts, J.K. Lenstra (Eds.), Local Search in Combinatorial Optimization, Wiley, Chichester, 1997, pp. 137–171.

[12] G. Schrimpf, K. Schneider, H. Stamm-Wilbrandt, V. Dueck, Record breaking optimization results using the ruin and recreate principle, J. Comput. Phys. 159 (2000) 139–171.

[13] N. Mladenović, P. Hansen, Variable neighbourhood search, Comput. Oper. Res. 24 (1997) 1097–1100.

[14] T. Koopmans, M. Beckmann, Assignment problems and the location of economic activities, Econometrica 25 (1957) 53–76.

[15] T.C. Hu, E.S. Kuh (Eds.), VLSI Circuit Layout: Theory and Design, IEEE Press, New York, 1985.

[16] J.W. Dickey, J.W. Hopkins, Campus building arrangement using TOPAZ, Transp. Res. 6 (1972) 59–68.

[17] A.N. Elshafei, Hospital layout as a quadratic assignment problem, Oper. Res. Q. 28 (1977) 167–179.

[18] E. Taillard, Comparison of iterative searches for the quadratic assignment problem, Location Sci. 3 (1995) 87–105.

[19] R.E. Burkard, J. Offermann, Entwurf von schreibmaschinentastaturen mittels quadratischer zuordnungsprobleme, Z. Oper. Res. 21 (1977) 121–132.

[20] R.E. Burkard, E. Çela, P.M. Pardalos, L. Pitsoulis, The quadratic assignment problem, in: D.Z. Du, P.M. Pardalos (Eds.), Handbook of Combinatorial Optimization, vol. 3, Kluwer, Dordrecht, 1998, pp. 241–337.

[21] S. Sahni, T. Gonzalez, P-complete approximation problems, JACM 23 (1976) 555–565.

[22] L.M. Gambardella, E. Taillard, M. Dorigo, Ant colonies for the quadratic assignment problems, J. Oper. Res. Soc. 50 (1999) 167–176.

[23] A. Bölte, U.W. Thonemann, Optimizing simulated annealing schedules with genetic programming, Eur. J. Oper. Res. 92 (1996) 402–416.

[24] E. Taillard, Robust taboo search for the QAP, Parallel Comput. 17 (1991) 443–455.

[25] C. Fleurent, J.A. Ferland, Genetic hybrids for the quadratic assignment problem, in: P.M. Pardalos, H. Wolkowicz (Eds.), Quadratic Assignment and Related Problems. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 16, AMS, Providence, 1994, pp. 173–188.

[26] M.H. Lim, Y. Yuan, S. Omatu, Efficient genetic algorithms using simple genes exchange local search policy for the quadratic assignment problem, Comput. Optim. Appl. 15 (2000) 249–268.

[27] P. Merz, B. Freisleben, A genetic local search approach to the quadratic assignment problem, Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA'97), East Lansing, USA, 1997.

[28] D.M. Tate, A.E. Smith, A genetic approach to the quadratic assignment problem, Comput. Oper. Res. 1 (1995) 73–83.

[29] R.E. Burkard, S. Karisch, F. Rendl, QAPLIB—a quadratic assignment problem library, J. Glob. Optim. 10 (1997) 391–403.

[30] E. Taillard, L.M. Gambardella, Adaptive memories for the quadratic assignment problem, Technical Report IDSIA-87-97, Lugano, Switzerland, 1997.