Computing and Informatics, Vol. 28, 2009, 1001–1012, V 2009-Mar-2

OPTIMIZATION OF QUADRATIC ASSIGNMENT PROBLEM USING SELF ORGANISING MIGRATING ALGORITHM

Donald Davendra, Ivan Zelinka

Department of Applied Informatics
Tomáš Baťa University
Nad Stráněmi 4511
760 01 Zlín, Czech Republic
e-mail: {davendrai, zelinka}@fai.utb.cz

Revised manuscript received 9 December 2008

Abstract. This paper introduces the primary research involving Self Organinsing Migrating Algorithm (SOMA) to the permutative problem of Quadratic Assignment. SOMA is transformed from its canonical form to successfully solve permutative optimization problems. Conversion and repairment routines are added to the generic SOMA. The results presented outline the high effectiveness of SOMA for solving QAP problems.

Keywords: Self organizing migrating algorithm, quadratic assignment problem, combinatorial optimization

1 INTRODUCTION

NP problem optimization comprises a large and important class of practical problems in science, engineering and business [1]. While real domain problems (non-linear, differential etc) comprise a large domain of optimization problems, permutative problems command a higher difficulty rating since most are intractable.

The principle application for such problems can be seen in real life applications, like manufacturing floor design and planning, routing problems like vehicle, network data amongst others. Layout planning and design alongside large scale manufacturing system depend on optimization tools [1].

Currently a number of heuristics exist which deal with these problems; however, most arose as canonically real optimization tools, later modified for permutative problems. Ant Colony [2], Differential Evolution [3, 4] are few of such heuristics.

This research introduces a next generation optimization heuristic of SOMA, which has proven highly effective in solving real optimization problems [5, 6]. SOMA is transformed to solve the Quadratic Assignment Problem (QAP) and its effectiveness is compared against some other optimization tools.

This paper is divided into the following sections. Section 2 outlines SOMA and its transformation routines. Section 3 describes QAP and its formulation. Section 4 discusses the experimentation and results and Section 5 concludes the work.

2 SELF ORGANINSING MIGRATING ALGORITHM

SOMA is a stochastic optimization algorithm modelled on the social networking behaviour of co-operating individuals [18]. It was chosen because it has been proven that the algorithm has the ability to converge towards the global optimum [5, 6].

SOMA works on a population of candidate solutions in loops called *migration loops* or generation loops. The population is initialized randomly which gives it a distribution over the search space at the beginning of the search. In each loop, the population is evaluated and the solution with the highest fitness becomes the leader. Apart from the leader, in one migration loop, all individuals will traverse the input solution space in the direction of the leader.

An individual will travel a certain distance (called the *path length*) towards the leader in n steps of defined length or *step size.* If the path length is chosen to be greater than one, then the individual will overshoot the leader. This path is perturbed randomly.

2.1 Perturbation

Mutation, the random perturbation of individuals, is an important operation for evolutionary strategies (ES). It ensures the diversity amongst the individuals and it also provides the means to restore lost information in a population. However, mutation is applied differently in SOMA compared with other ES strategies. SOMA uses a parameter called PRT to achieve perturbation. This parameter has the same effect for SOMA as mutation has for GA. It is defined in the range $[0, 1]$ and is used to create a perturbation vector (PRT Vector) as follows:

$$\begin{array}{l}\text{if } rnd_j < PRT \text{ then } PRTVector_j = 1 \\ \text{else } 0, \quad j = 1, \dots, n\end{array} \tag{1}$$

The novelty of this approach is that the PRT Vector is created before an individual starts its journey over the search space. The PRT Vector defines the final movement of an active individual in search space.

The randomly generated binary perturbation vector controls the allowed dimensions for an individual. If an element of the perturbation vector is set to zero, then the individual is not allowed to change its position in the corresponding dimension.

2.2 Generating New Candidate Solutions

In standard ES the *Crossover* operator usually creates new individuals based on information from the previous generation. Geometrically speaking, new positions are selected from an N dimensional hyper-plane. In SOMA, which is based on the simulation of cooperative behaviour of intelligent beings, sequences of new positions in the N-dimensional hyperplane are generated. They can be thought of as a series of new individuals obtained by the special crossover operation. This crossover operation determines the behaviour of SOMA. The movement of an individual is thus given as follows:

$$\vec{r} = \vec{r}_0 + \vec{m}tPRTVector \qquad (2)$$

where:

- $\vec{r}$: new candidate solution
- $\vec{r}_0$: original individual
- $\vec{m}$: difference between leader and start position of individual
- $t \in [0, \text{Path length}]$
- *PRTVector*: control vector for perturbation.

It can be observed from Equation (2) that the PRT vector causes an individual to move toward the leading individual (the one with the best fitness) in $N - k$ dimensional space. If all N elements of the PRT vector are set to 1, then the search process is carried out in an N dimensional hyperplane (i.e. on a $N + 1$ fitness landscape). If some elements of the PRT vector are set to 0 then the second term on the right-hand side of Equation (2) equals to 0. This means those parameters of an individual that are related to 0 in the PRT vector are 'frozen', i.e. not changed during the search. The number of frozen parameters k, is simply the number of dimensions that are not taking part in the actual search process. Therefore, the search process takes place in an $N - k$ dimensional subspace.

2.3 Permutative Approach

As mentioned earlier, the canonical approach of SOMA is for differential problem, which accepts real numbers as parameters. In this research, SOMA is slightly modified to work within the integer domain, specifically permutative. Permutative implies that each unique value in the solution.

The process to transform the heuristic is done in two parts:

1. permutative population
2. repairment of solution.

2.3.1 Permutative Population

As in the general case with all heurisitcs, the initial population is randomly generated to resemble a permutative structure. For example, if solving a 5 size problem, a solution string containing 5 unique integers ranging from 1 to 5 will be randomly generated.

2.3.2 Repairment of Solution

SOMA transforms the solution into real values, simply through its inner transformation. The resulting solution is real. In order to transform it back into the integer domain, a simple rounding of the value is done to achieve its integer equivalent as given in the following equation:

$$\vec{r}_i = \mathrm{int}\left[\vec{r}_i\right]. \tag{3}$$

Subsequently, the entire solution is rendered integer. At this point, a repairment procedure is applied to first bring all values within the operating range, and then a repairment procedure is applied.

$$\text{upper bound } \leq \vec{r}_i \leq \text{ lower bound} \tag{4}$$

2.4 Repairment Pseudocode

The first routine scans the solution for all the missing values. Starting from an index in the solution, all subsequent values are scanned for replicated value. If one is detected, then it is stored in the missing value array.

```
if (solution is infeasible)
Scan solution for missing values
For ($i = 1; i <$ Upperbound; $i$++) {
  Bool = false;
  For ($j = 0; j <$ Upperbound; $j$++) {
    If (Solution == $i$)
    Bool = true;
  }
If (Bool == false)
  missing val array = $i$
}
```

The second routine assigns a value 0, for all positions which contain a replicated value. This simplifies the potions for replacement.

```
*Scan solution for replicated values
For ($i = 1; i <$ Upperbound; $i$++) {
   Pos Array = 0;
   For ($j = 0; j <$ Upperbound; $j$++) {
      If (Solution == $i$)
      Pos Array = 0;
   }
If (Pos Array $> 1$)
   Replicated pos = Pos Array
}
```

The third routine replaces the values assigned to 0 by a value from the missing value array.

```
*Randomly select values to be replaced with missing values
For ($i = 0; i <$ size of missing value array; $i$++)
   Rand [Solution$_{\text{Replicated pos}}$] = Rand [missing val array]
   Delete value from Replicated Pos array
   Delete Rand [missing val array]
```

The last function is presented in simplified format. In essence it shows that a random value is selected from the *missing val array*, and this value is inserted in the solution array indexed by a value randomly selected from the *replicated pos array*. After completion, both these values are deleted from their respective arrays.

The primary reason for demonstrating this routine is that it has proven to be highly effective in repairing infeasible solutions [7, 8]. However, the reader should keep in mind that only infeasible solution are repaired. The repaired solution is then tested for its fitness value, and compared against the leader of the population.

A schematic of the operations of SOMA is given as an example in Figures 1–3.

Figure 1 gives the initial conditions of SOMA. For illustration purpose, the operating parameters in Figure 1 are set in "Parameter Settings". The Step Size is 0.23, Path Length is 3 and PRT is set to 0.1.

The crossover and mutation schema of SOMA is illustrated in Figure 2. Only 5 solutions are generated, and vetted for its fitness. As shown, Solution 3 has the minimal fitness, and is selected as the "leader".

Taking each solution in turn, a jump sequence is done towards the leader using the SOMA routine. However, the obtained solutions can be infeasible (real values) as shown by value 2.92. The "repair" routine repairs the solution (from 2.92 to 5) so that all values are permutative.

The new fitness of the solution is calculated, and if it improves on the old fitness as is the case of the third solution with value 34 231, the new soluion replaces the old

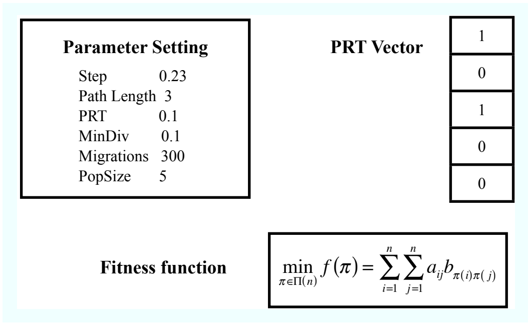


Fig. 1. Initial Conditions

solution with value 65 376 in the "population". After each iteration, the "leader" is recalculated.

3 QUADRATIC ASSIGNMENT PROBLEM

The QAP is a combinatorial optimization problem stated for the first time by [9] and is widely regarded as one of the most difficult problems in this class. The approach is to have two matrices of size $n \times m$ given as:

$$A = (a_{ij}) \tag{5}$$

$$B = (b_{ij}) \tag{6}$$

The objective is then to find the permutation π^* which minimizes

$$\min_{\pi \in \prod(n)} f(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{\pi(i)\pi(j)} \tag{7}$$

where $\prod(n)$ is a set of permutations of n elements. QAP is considered a *NP* hard problem [14] and problem sizes larger than 20 are considered intractable. Many application have been identified for QAP, which include, amongst others, the allocation of plants to candidate locations; layout of plants; backboard wiring problem; design of control panels and typewriter keyboards; balancing turbine runners; ordering of interrelated data on a magnetic tape; processor-to-processor assignment in a distributed processing environment; placement problem in VLSI design; analyzing chemical reactions for organic compounds; and ranking of archaeological data. The details and references for these and additional applications can be found in [11].

For large sized problems, heuristic approaches have been developed. The most notable ones include: simulated annealing [12], tabu searches of [11, 12, 13], the hybrid genetic-tabu searches [14] and the ant colony approach [2].

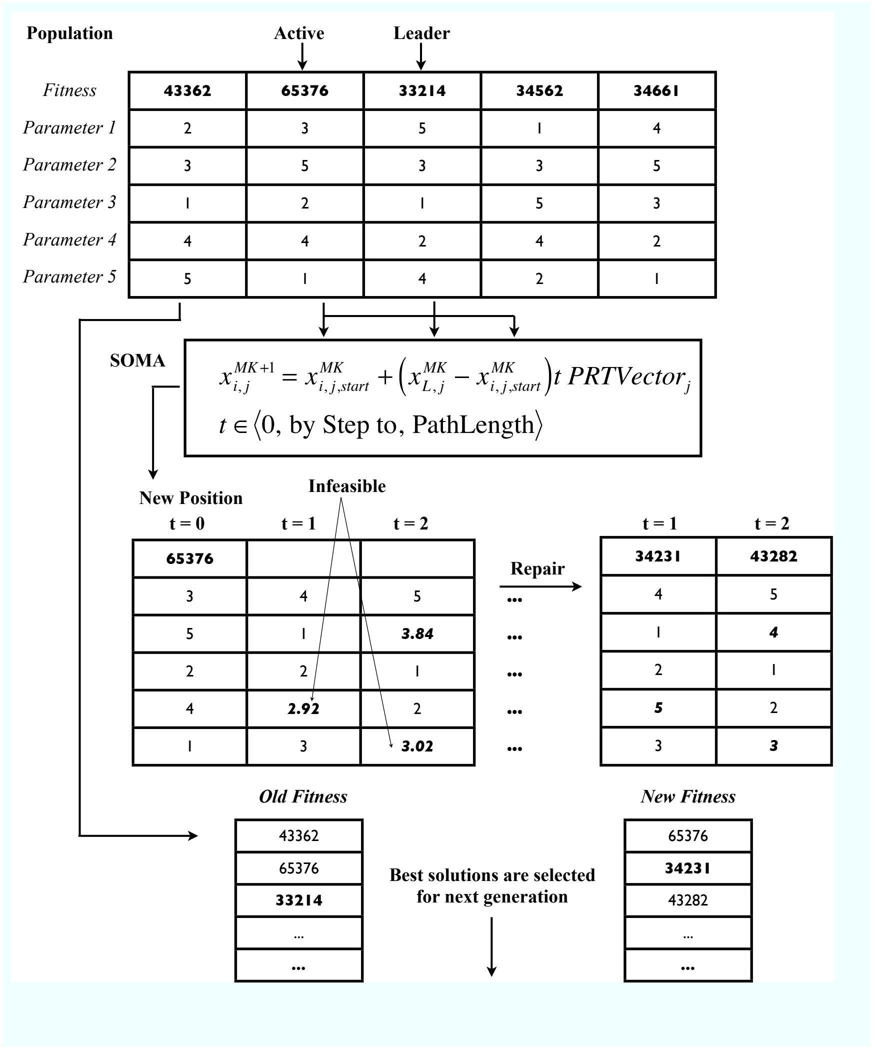
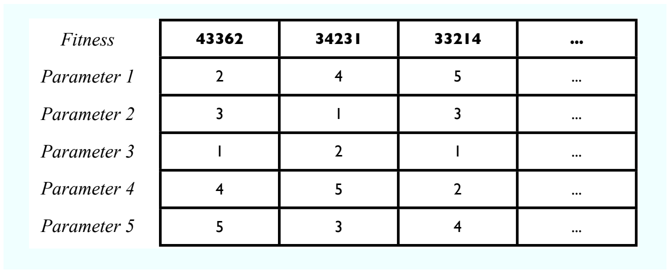

Population
Active
Leader
Fitness
Parameter 1
Parameter 2
Parameter 3
Parameter 4
Parameter 5
43362
65376
33214
34562
34661
SOMA
$x_{i,j}^{MK+1} = x_{i,j,start}^{MK} + \left(x_{L,j}^{MK} - x_{i,j,start}^{MK}\right) t \; PRTVector_j$
$t \in \langle 0, \text{by Step to, PathLength} \rangle$
New Position
Infeasible
t = 0
t = 1
t = 2
2.92
3.84
3.02
Repair
34231
43282
Old Fitness
New Fitness
Best solutions are selected for next generation

Fig. 2. SOMA schematic for QAP

4 RESULTS AND ANALYSIS

The benchmark QAP problems were obtained from the OR Library [15] which is the respository of all benchmark scheduling problems. Two different sets of experimentations were conducted; on *Irregular* and *Regular* problems.

The difference between regular and irregular problems is based on the $flow - dominance(fd)$ which is used to differentiate among the classes of QAP instances. It is defined as a coefficient of variation of the flow matrix entries multiplied by 100.



Fitness	**43362**	**34231**	**33214**	**...**
Parameter 1	2	4	5	...
Parameter 2	3	1	3	...
Parameter 3	1	2	1	...
Parameter 4	4	5	2	...
Parameter 5	5	3	4	...

Fig. 3. Final Solution

$$
\begin{aligned}
&fd = \tfrac{100\sigma}{\mu} \\
&\text{where:} \\
&\mu = \tfrac{1}{n^2} \cdot \textstyle\sum_{i=1}^{n} \sum_{j=1}^{n} b_{ij} \\
&\sigma = \sqrt{\tfrac{1}{n^2-1} \cdot \textstyle\sum_{i=1}^{n} \sum_{j=1}^{n} (b_{ij} - \mu)}
\end{aligned}
\qquad (8)
$$

Irregular problems have *fd* larger than 1.2 where as regular problems have *fd* lower than 1.2.

The results produced by SOMA for these two different problem sets are given in Tables 1 and 2. These are the average values over 10 iterations, each having 200 migrations.

Prob	FD	n	Optimal	TT	RTS	SA	GH	HAS	SOMA
bur26a	2.75	26	5 246 670	0.208	–	0.1411	0.012	0	0
bur26b	2.75	26	3 817 852	0.441	–	0.1828	0.022	0	0
bur26c	2.29	26	5 426 795	0.17	–	0.0742	0	0	0
bur26d	2.29	26	3 821 225	0.249	–	0.0056	0.002	0	0
bur26e	2.55	26	5 386 879	0.076	–	0.1238	0	0	0
bur26f	2.55	26	3 782 044	0.369	–	0.1579	0	0	0.03
bur26g	2.84	26	10 117 172	0.078	–	0.1688	0	0	0
bur26h	2.84	26	7 098 658	0.349	–	0.1268	0.001	0	0
chr25a	4.15	26	3 796	15.969	16.85	12.497	2.692	3.082	0.129
els19	5.16	19	17 212 548	21.261	6.714	18.538	0	0	0
kra30a	1.46	30	88 900	2.666	2.155	1.4657	0.134	0.629	0.002
kra30b	1.46	30	91 420	0.478	1.061	1.065	0.054	0.071	0.03
tai20b	3.24	20	122 455 319	6.7	–	14.392	0	0.091	0.004
tai25b	3.03	25	344 355 646	11.486	–	8.831	0	0	0
tai30b	3.18	30	637 117 113	13.284	–	13.515	0.001	0	0.043
tai35b	3.05	35	283 315 445	10.165	–	6.935	0.107	0.026	0
tai40b	3.13	40	637 250 948	9.612	–	5.43	0.211	0	0.02
tai50b	3.10	50	458 821 517	7.602	–	4.351	0.212	0.192	0.2

Table 1. Comparison data of SOMA for regular QAP problems

Prob	FD	n	Optimal	TT	RTS	SA	GH	HAS	SOMA
nug20	0.99	20	2 570	0	0.911	0.070	0	0	0
nug30	1.09	30	6 124	0.032	0.872	0.121	0.007	0.098	0.020
sko42	1.06	42	15 812	0.039	1.116	0.114	0.003	0.076	0.010
sko49	1.07	49	23 386	0.062	0.978	0.133	0.040	0.141	0.005
sko56	1.09	56	34 458	0.080	1.082	0.110	0.060	0.101	0.010
tai20a	0.61	20	703 482	0.211	0.246	0.716	0.628	0.675	0
tai25a	0.60	25	1 167 256	0.510	0.345	1.002	0.629	1.189	0
tai30a	0.59	30	1 818 146	0.340	0.286	0.907	0.439	1.311	0.010
tai35a	0.58	35	2 422 002	0.757	0.355	1.345	0.698	1.762	0.030
tai40a	0.60	40	3 139 370	1.006	0.623	1.307	0.884	1.989	0.623
tai50a	0.60	50	4 941 410	1.145	0.834	1.539	1.049	2.800	0.645
wil50	0.64	50	48 816	0.041	0.504	0.061	0.032	0.061	0.070

Table 2. Comparison data of SOMA for irregular QAP problems

For these experimentations, the SOMA version of **All−to−One** was selected. The reasoning for this issue was the issue of optimizing the execution time. The operations parameters of SOMA as outlined in Section 2 were found through experimentation and are given in Table 3.

PopSize	Migration	PathLength	Step Size	PRT
200	200	3.0	0.23	0.1

Table 3. Operational parameters for SOMA

Three native SOMA parameters are to be optimized, namely *PathLength*, *Step Size* and *PRT*. *PRT* is chosen as 0.1 therefore a maximum of 90 % of the dimensions are under consideration at any iteration. The *PathLength* is set as 3, since the solution jumps two steps beyond the leader from its starting point. *PathLength* longer than 3 is not deemed feasible due to the extended execution time required. The *StepSize* is the only variable which can generally be said to be problem dependent. The starting point of 0.1 and incrementation of $0.01 - 0.02$ per execution would be sufficient to obtain the optimal operating value. Population size and number of iterations/migrations are, as in all metaheuristics, problem dependent.

Comparisons were made with other heuristics of Tabu Search TT [11], Reactive Tabu Search (RTS) [16] and the Genetic Hybrid (GH) method [14]. A Simulated Annealing (SA) [12] approach that is cited as a good implementation in [17] was also included. Finally the work covered [18] with Ant Colony (HAS-QAP) is compared as the best results for these instances of QAP.

The average quality of the solutions produced by the methods is shown, measured in per cent above the best solution value known from the OR Library [15]. The best results obtained are indicated in boldface. The well performing heuristics are able to produce solutions with at less than 1 % with the same computing power and time.

Analysis of the results reveals that SOMA functions exceptionally well in the both classes of problems. For the *bur* problem class, it only fails to find one optimal solution, and is the best heuristic in the *chr* problem. It performs better in the *kra* problem set and competes evenly in the *tai* set with the other heuristic of HAS-QAP. On reflection HAS-QAP only performs better on 4 instances than SOMA.

The irregular problems reflect the advantage of SOMA more clearly. SOMA obtains the best solution in all but two problem instances of *sko42* and *wil50*, and the optimal solution in the three instances of *nug20*, *tai20* and *tai25*.

A sample generation output is given in Figure 4.

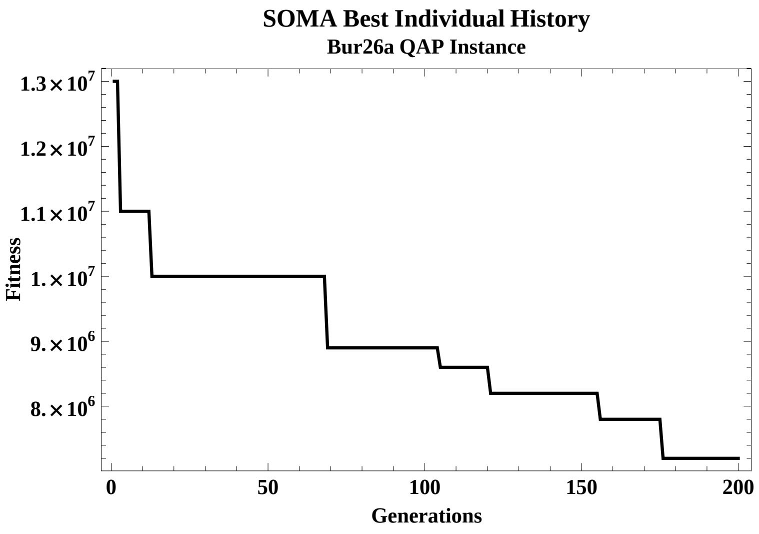


Fig. 4. QAP generation output

5 CONCLUSION

The application domain of SOMA has been extended in this research to permutative NP problems. The experimentations with QAP have validated the stated approach, with comparable results, both with other heuristics but more importantly with the optimal solutions. The results obtained especially for the irregular problems highlighted the effectiveness of SOMA.

The authors believe that this is the first application of SOMA in NP problems and further research would entail the application to other problems like flow shop and traveling salesman.

Acknowledgment

This research was supported by grant from the Grant Agency of Czech Republic GACR 102/06/1132.

REFERENCES

[1] Onwubolu, G.: Emerging Optimization Techniques in Production Planning and Control. Imperial College Press, London, 2002.

[2] Dorigo, M.—Gambardella, L.: Ant Colony System: A Co-operative Learning Approach to the Traveling Salesman Problem. IEEE Transaction on Evolutionary Computations, 1997, No. 1, pp. 53–65.

[3] Price, K.: An Introduction to Differential Evolution. In: D. Corne, M. Dorigo and F. Glover (Eds.): New Ideas in Optimization, McGraw-Hill, London, UK, 1999, pp. 79–108.

[4] Onwubolu, G.—Davendra, D.: Scheduling Flow Shops Using Differential Evolution Algorithm. European Journal of Operations Research, Vol. 171, 2006, pp. 674–679.

[5] Zelinka, I.: SOMA – Self Organizing Migrating Algorithm. In: G. Onwubolu and B. Babu, (Eds.): New Optimization Techniques in Engineering, Springer-Verlag, 2004.

[6] Zelinka, I.—Lampinen, J.—Nolle, L.: On the Theoretical Proof of Convergence for a Class of SOMA Search Algorithms. Proceedings of the 7th International MENDEL Conference on Soft Computing, Brno, CZ, June 6–8, 2001, pp. 103–110.

[7] Davendra, D.: Hybrid Differential Evolution Algorithm for Discrete Domain Problems. Master of Science Thesis, University of the South Pacific, Suva, Fiji Islands, 2003.

[8] Garca, P.—Gonzalez-Castano, F.—Burguillo-Rial, J.: A Combined Global & Local Search (CGLS) Approach to Global Optimization. Journal of Global Optimization, Vol. 34, 2006, No. 3, pp. 409–426.

[9] Koopmans, T.—Beckmann, M.: Assignment Problems and the Location of Economic Activities. Econometrica, 1957, No. 25, pp. 53–76.

[10] Sahni, S.—Gonzalez, T.: P-Complete Approximation Problems. Journal of the ACM. 1976, No. 23, pp. 555–565.

[11] Battitti, R.—Tecchiolli, G.: The Reactive Tabu Search. ORCA Journal on Computing, Vol. 6, 1994, pp. 26–140.

[12] Connolly, D.: An Improved Annealing Scheme for the QAP. European Journal of Operation Research, 1990, No. 46, pp. 93–100.

[13] Taillard, E.: Robust Taboo Search for the Quadratic Assignment Problem. Parallel Computing, Vol. 17, 1991, pp. 443–455.

[14] Fleurent, C.—Ferland, J.: Genetic Hybrids for the Quadratic Assignment Problem. Operations Research Quarterly, Vol. 28, 1994, pp. 167–179.

[15] OR Library web site. Available at: `http://mscmga.ms.ic.ac.uk/info.html`.

[16] Taillard, E.: Comparison of Iterative Searches for the Quadratic Assignment Problem. Location Science, 1995, No. 3, pp. 87–105.

[17] Burkard, R.: Location With Spatial Interactions: The Quadratic Assignment Problem. In: P. Mirchandani and R. Francis (Eds.): Discrete Location Theory, John Wiley, 1991.

[18] Gambardella, L.—Thaillard, E. D.—Dorigo, M.: Ant Colonies for the Quadratic Assignment Problem. Journal of Operational Research, 1999, No. 50, pp. 167–176.

Donald Davendra is a Ph. D. candidate in technical cybernetics at Tomáš Baťa University in Zlín, Czech Republic. He has a M. Sc. in computer science from the University of the South Pacific, Fiji. His research area is in intelligent manufacturing and chaotic systems applied to evolutionary heuristics.

Ivan Zelinka is the Associate Professor of Informatics at Tomáš Baťa University in Zlín. He is the recipient of the Siemens Best European Ph. D. Dissertation Award for his work on SOMA. He is also the inventor of analytical programming, and author of books and articles on chaotic and evolutionary systems. His website is `www.fai.utb.cz/people/zelinka/hp`.