The team priority assignment problem

Matthias Lohmann, Manuel Molina Madrid,
Daniel Lückerath, Ewald Speckenmeyer
Institut für Informatik, Universität zu Köln,
Albertus-Magnus-Platz, 50923 Köln, Germany

Oliver Ullrich
National Science Foundation's
Industry-University Cooperative Research Center
for Advanced Knowledge Enablement,
Florida International University,
11200 SW 8th St, ECS 354, FL 33199 Miami

September 10, 2015

1 Introduction

Like many other university courses, University of Cologne's (UoC) programming class can be attended by differently sized teams (of typically two to six students), who handle presentations and homework collectively. The course consists of up to 30 work groups with different capacities taking place at different times and days of the week. The organizers found it desirable to enable to students who formed a team prior to the course to participate together with their peers in the same work group. Because students typically have tight schedules, it is preferable to give individual students the opportunity to weight the available time slot options according to their individual priorities. In this process students should also be able to put an individual weight on their collaboration with their established team. If the marginal benefit of staying with this team is outweighed by the cost of being collectively assigned to a non-optimum work group, the team should be split, and more favorable assignments of the individual students should be generated. Finding an optimum team priority assignment of student teams to work groups is related to the Generalized Assignment Problem (GAP, see [9], pp. 189-220), which is well known to be NP-hard.

This paper presents the Team Priority Assignment Problem (TPAP), describes its relationship to GAP and shows that TPAP as a combinatorial optimization problem is NP-hard. A method based on the simulated annealing meta heuristic (see [4]) is presented, which enables the course organizers to generate high-quality - though not optimal - solutions in reasonable run time for team priority assignments of UoC's programming class.

This report continues by sharing some background on the team priority assignment problem and recent research on that subject (section 2). Following on that, a heuristic approach to the problem is presented (section 3). Some experiments are conducted, demonstrating the approach's suitability to provide high quality solutions to real-world instances (section 4). The paper closes with a short summary of lessons learned and some thoughts on further research (section 5).

2 Background

2.1 Team priority assignment

The objective of TPAP is to find optimum assignments $x_{i,j} \in \{0,1\}$ of students $s_i \in S$ with $n := |S|$ to work groups $g_j \in G$, $m := |G|$ with a capacity μ_j. Here, $x_{i,j}$ is set to 1 if student s_i is assigned to group g_j, or set to 0 if not. Each student s_i awards a weight $c_{i,j} \in \mathbb{R}^+$ for each possible assignment to a group g_j. She also indicates the team t_i she is part of, and sets a team priority $p_i \in \mathbb{R}^+$ which represents the value of being assigned to the same group as the other members of team t_i.

The objective function $f(x)$ consists of two parts $g(x)$ and $h(x)$. If all team members are assigned to the same work group, no further cost are added to the global sum of assignment weights $g(x)$. If this is the case for all students, the objective function consists only of the combined group assignment cost shown in equation 1.

$$g(x) = \sum_{i=1}^{n}\sum_{j=1}^{m}(x_{i,j} \cdot c_{i,j}) \tag{1}$$

If not all members of a team t_j are assigned to the same work group, the team priority values p_i for all s_i with $t_i = t_j$ become effective according to equation 2.

$$h(x) = \sum_{i=1}^{n}\sum_{j=1}^{m} x_{i,j} \cdot b_{i,j} \cdot p_i \tag{2}$$

The factor $b_{i,j}$ is calculated in the following way (see equation 3): If the set T_{t_i} of students in team t_i consists only of one student (namely student s_i), $b_{i,j}$ is set to 0. If T_{t_i} consists of two or more students, $b_{i,j}$ is calculated as the portion of the team which is not assigned to the work group containing student s_i taken to the power of four. The fourth power was chosen because with teams of the expected sizes the resulting values discourage separating a single student

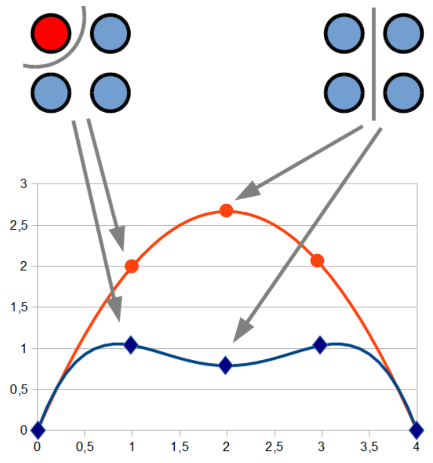


Figure 1: Cost resulting from separating various numbers of students from their team with power-of-four (blue) and linear (red) objective function component

from her team (see figure 1 for a team size of four).

$$b_{i,j} = \begin{cases} \left(\frac{\sum_{s \in T_{t_i}} (1 - x_{s,j})}{|T_{t_i}| - 1}\right)^4 & \text{for } |T_{t_i}| > 1 \\ 0 & \text{for } |T_{t_i}| = 1 \end{cases} \tag{3}$$

If a team has to be split, it is thus encouraged to be split in equal parts. This maps the desire not to isolate single students but to keep at least partial teams intact.

Building up on this, TPAP as a combinatorial optimization problem can be defined as finding an assignment matrix x with $x_{i,j} \in \{0, 1\}$, which minimizes the objective function $f(x)$, combined from $g(x)$ and $h(x)$, and shown in equation 4.

$$f(x) = g(x) + h(x) = \sum_{i=1}^{n} \sum_{j=1}^{m} x_{i,j} \cdot (c_{i,j} + b_{i,j} \cdot p_i) \tag{4}$$

A valid solution has to adhere to two restrictions (see table 1): restriction (a) guarantees that the maximum size μ_j is observed for each group j; restriction (b) defines that each student is assigned to exactly one group.

2.2 Computational complexity

To decide on TPAP's computational complexity we show in a first step that special cases of the Generalized Assignment Problem (GAP) can be reduced to

$$\begin{array}{lll}
\text{Minimize} & f(x) = \sum_{i=1}^{n} \sum_{j=1}^{m} x_{i,j} \cdot (c_{i,j} + b_{i,j} \cdot p_i) & \\
\text{with} & b_{i,j} = \begin{cases} (\frac{\sum_{s \in T_{t_i}} (1 - x_{s,j})}{|T_{t_i}| - 1})^4 & \text{for } |T_{t_i}| > 1 \\ 0 & \text{for } |T_{t_i}| = 1 \end{cases} & \\
\text{and} & T_t = \{i \in \{1, ..., n\} | t_i = t\} & \forall t \in \mathbb{N} \\
\text{Subject to} & & \\
\text{(a)} & \sum_{i=1}^{n} x_{i,j} \leq \mu_j & \forall j \in \{1, ..., m\} \\
\text{(b)} & \sum_{j=1}^{m} x_{i,j} = 1 & \forall i \in \{1, ..., n\} \\
 & x_{i,j} \in \{0, 1\} &
\end{array}$$

Table 1: The TPAP optimization problem

TPAP. 3-Partition, which is known to be NP-complete, is then reduced to these GAP cases. It is thus shown that TPAP is NP-complete, or - as an optimization problem - NP-hard.

As described, TPAP is closely related to a special case of GAP, where usually terms of production and logistics (e.g. tasks, machines, a machine's capacity) are applied. GAP instances can here be interpreted as assignments of student teams (i.e. tasks) to work groups (i.e. machines), in which teams cannot be split (unlike in TPAP). As in TPAP, different assignments result in different cost for the whole team. Also as in TPAP, GAP work groups can have different capacities, and team sizes can vary. Unlike TPAP, not every instance of GAP has a feasible solution, and team sizes can be dependent on the group a team is assigned to. This is analogous to GAP's tasks being assigned to machines of different capacity and throughput, where team sizes represent different necessary machine times on different machines. In short: GAP is more flexible than TPAP in allowing different team sizes depending on the assigned work group, but less flexible in that the teams cannot be split up.

Otherwise, the problems are similar: For instances of TPAP which have their team priorities set to infinity, finding a result of finite cost is equivalent to solving a GAP instance in which TPAP's individual students' cost are combined to team cost. If a GAP instance has no feasible solution, all solutions of the TPAP instance have infinite cost. In reverse, an instance of GAP in which team sizes do not change depending on the assigned work group can be seen as an instance of TPAP. This is formed by

1. deconstructing every team into single students,

2. setting their $c_{i,j}$ so that for each group the assignment of all team members to this group yields the same cost as assigning the team to the equivalent group in GAP, and

3. setting their team priorities to infinity.

It can now be shown that TPAP as a combinatorial optimization problem is NP-hard. To accomplish this, we show that 3-Partition (a well-known NP-complete problem, see [5]) can be reduced to TPAP. A 3-Partition instance consists of a multiset (a set in which members can appear more than once) $A = \{a_1, \dots a_{3m}\}$, with $a_i \in \mathbb{N}$ and $B \in \mathbb{N}$ with $B \cdot m = \sum_{i=1}^{3m} a_i$. The problem is defined as finding a partition of A into m subsets so that for each set, the sum of its members equals B. This is NP-complete even if $B/4 < a_i < B/2$, which forces each subset to contain exactly three members.

Given a 3-Partition instance, an equivalent GAP instance is created as follows (analogous to [16], pp. 801-802):

1. Create $3m$ teams, where team i has a constant size of a_i students regardless of the assigned group, for all $i \in 1, \dots, 3m$,

2. set all assignment cost to the same arbitrary value, and

3. create m work groups of size B.

Each solution of this GAP instance corresponds to a solution of the 3-Partition instance: For each group, the assigned teams correspond to a set of integers of the 3-Partition instance whose sum is B. This GAP instance can be mapped to a TPAP instance as described above.

2.3 Related research

Several approaches on the general assignment problem and its derivatives exist, including [3], [6], [11], [12], [13], [14], [15] and [16] (for an overview which also includes some extensions, see [2]). Several of those methods are based on exact algorithms, like branch-and-bound in [8] and [14], or branch-and-prize in [15].

Many special cases of GAP are well-known problems of their own, e.g. the multiple knapsack problem (see [9], pp. 157-188, for the regular knapsack problem also see [9], pp. 13-80), or the linear sum assignment problem (see [1], pp. 4-7). Techniques for these special cases include the Hungarian Method for the linear sum assignment problem (see [6], improved in [11]), which can be implemented with cubic time complexity, and bound-and-bound for the multiple knapsack problem (see [9], pp. 170-176).

Other authors apply heuristic methods, like tabu search in [3] and [12], simulated annealing in [12], or a combination of methods in [13].

Some authors, e.g. [7] and [16], regard the assignment of military personnel to certain positions considering several policies as the Personnel Assignment Problem (PAP). Some variants of PAP are very similar to TPAP, in that they consider suitability of personnel for positions; [16] also tries to prevent relocating e.g. married couples to different military bases. Different from TPAP, [16] also includes hierarchical-ordering constraints, i.e. the military ranks of the personnel should match the hierarchy of the assigned positions. [16] applies an genetic algorithm to some of those sub-problems, including one called “Assignment Problem with Set Constraints” (APSC). Apart from the described differences, APSC has strong similarities with TPAP, especially its definition of set constraints which partition personnel into subsets which must be assigned as a whole to a single subset of positions. This corresponds to teams being assigned to the same work group in TPAP.

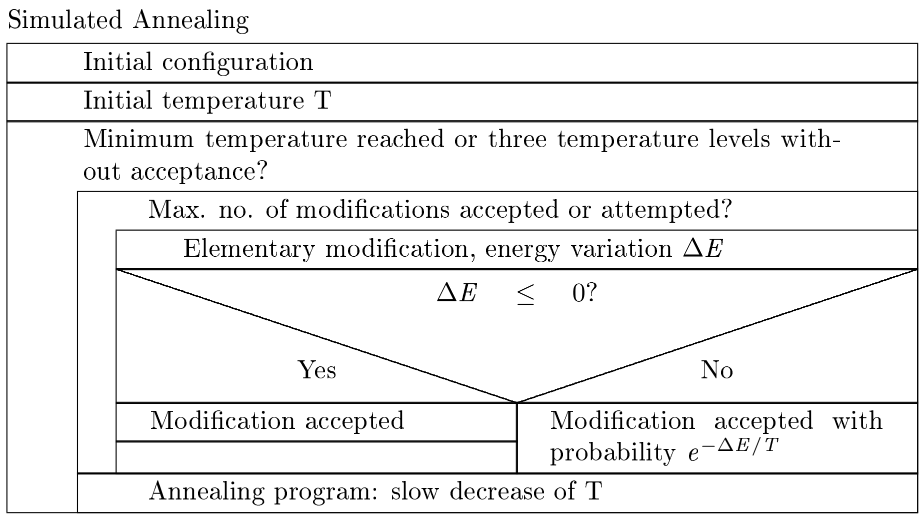
3 A heuristic approach to team priority assignment

To find good solutions in a reasonable time, a heuristic method is presented based on the simulated annealing technique (see [4]), which emulates the physical process of heating and controlled cooling of material to increase the regularity of its crystals. Similar to this process, the simulated annealing method (for an overview see figure 2) attempts to “cool down” the instance in a controlled way to find a good compromise of exploring both the objective function landscape globally, and researching local optima. The method initializes with a given starting temperature T_0 and a valid, but randomly constructed solution candidate $a = a_{approx}$, which assigns each student to a work group.

For each temperature level T_i an inner loop is executed: In each of its steps, an elementary modification is considered based on the current solution candidate a, which consists of the exchange of two students' assignments. The resulting new solution candidate a_n is either accepted as $a := a_n$ or rejected. The candidate is accepted if either $f(a_n) \leq f(a)$, or with a probability corresponding to the Metropolis rule (see [10]), i.e. if $p < e^{(\frac{-(\triangle E)}{T_i})}$, with a random $p \in [0, 1)$ and $\Delta E = f(a_n) - f(a)$. If an accepted candidate a has a lower objective function value than a_{approx}, a is accepted as the new approximate solution $a_{approx} := a$.

This inner loop is executed until a maximum number of modifications is conducted or a maximum number of acceptances is reached. Then, the temperature level is lowered. The outer loop is executed until a minimum temperature is reached or three succeeding temperature stages were executed without any acceptances.

The set of necessary parameters consists of a starting temperature T_0, the maximum number of modifications on a temperature level, the law of temperature change between stages, and a condition for the algorithm's termination. [4] recommends a set of parameters which is applied as a starting point for further calibration.



Simulated Annealing
Initial configuration
Initial temperature T
Minimum temperature reached or three temperature levels without acceptance?
Max. no. of modifications accepted or attempted?
Elementary modification, energy variation ΔE
$\Delta E \leq 0$?
Yes
No
Modification accepted
Modification accepted with probability $e^{-\Delta E/T}$
Annealing program: slow decrease of T

Figure 2: The simulated annealing meta heuristic (as described in [4])

4 Experiments

As described, the motivation for this work is to generate high quality solutions for a real-world instance of TPAP, the students' assignment to work groups for University of Cologne's programming class. To assess the behavior of the proposed model and the heuristic solver, we begin by experimenting on randomly generated instances from 100 to 1000 students. Following on that, the students' assignment of the 2013/14 programming class is examined.

4.1 Random instances

Input data for test instance generation consist of the number n of students, average team size z, and a maximum group size g. The maximum number of teams is calculated as $k = \lceil \frac{n}{z} \rceil$, the number of groups as $m = \left\lceil \frac{n}{g} \right\rceil$. To construct an instance n students are generated. For each student s_i an assignment t_i to a team is picked from an uniform distribution. To map the preferences of student s_i, both the team priority $p_i \in [0, 1)$ and random assignment weights $\mu_{i,j} \in [0, 1)$ for each group j are also picked from a uniform distribution.

For the experiments, instances from 100 up to 1000 students are generated; the maximum group size is set to 20, the average team size is set to 3, which fits our observations concerning students' preferences for University of Cologne's programming class (see section 4.2).

The parameter set for simulated annealing is constructed as suggested in [4]: The initial temperature T_0 is calculated as $T_0 = \frac{-\Delta E}{\ln 0.5}$, with ΔE computed by averaging the absolute change in the objective function in 100 random perturbations a_i (see equation 5).

$$\Delta E = \frac{\sum_{i=1}^{100} |f(a) - f(a_i)|}{100} \qquad (5)$$

The change in temperature between stages is set to $T_{i+1} = 0.9 * T_i$. The algorithm lowers the temperature when $12 * N$ perturbations were accepted or $100 * N$ perturbations were attempted (where we set the chain length to $N = 100 * n$, which yields a better solution quality than setting $N = n$ as suggested in [4]). The algorithm is terminated if three succeeding stages without any acceptance occur, or a minimum temperature of $T_{min} = 0.00001$ is reached. Thirty runs are executed for each measuring point.

4.2 The University of Cologne's programming class

The test instance consists of 755 students, 405 of which prefer to work in one of 127 teams, 350 students prefer to work alone. The average team size for teams of more than one student is 3.19, with a minimum of two and a maximum of 13 students. The teams have to be assigned to 28 work groups, of which 13 groups have a maximum size of 33, and 15 have a maximum size of 22. The combined capacity of these groups is 759, they thus include four spare slots. As described, students were free to weight each prospective assignment and to set a team priority p_i. To accomplish an equal consideration of each student and thus to discourage "cheating" by setting very high weights, the combined sum of all weights for a student is set to 1000.

The parameter set for simulated annealing is again adapted following the suggestions in [4], as described in section 4.1.

4.3 Results and discussion

The results of the computation of random instances are shown in table 2 and figure 3. As expected, team cost, group cost and combined cost rise linear with the size of the instances. The algorithm's run time also rises as expected; the rise corresponds to the rise of the possible number of perturbations (between $12*N$ and $100*N$) on each temperature level. At an instance size of 800 (which is about the size of the real-world example), it amounts to 20.5 minutes. At this size, 190 (23.8%) out of 800 students were assigned to their first priority work group, 227 (28.4%) students to their second and third priority, and 383 (47.9%) students to a priority of four or more. Out of 222 teams of two or more students, 108 (48.7%) were assigned to one single work group, 107 (48.2%) were split up to two partial teams, 7 (3.2%) were split up to three parts. Only 73 students (9.1%) were assigned to a work group without any team mates.

The real-world instance is computed thirty times. The algorithm yields a best solution with an overall cost of 1,757.9 (average: 2,196.9), group cost of 1,610 (average: 1,998.4), and team cost of 147.9 (average: 198.5). The standard deviation of the thirty runs lies at 175.4 ($\frac{\sigma}{avg.\ cost} = 0.080$). The average run time for this instance is 1,211.6 seconds (20.2 minutes).

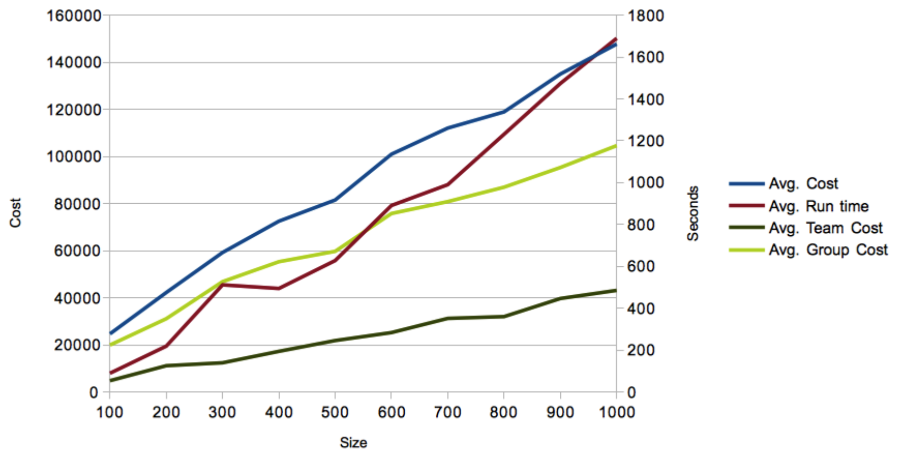


Figure 3: Cost and run time for random instances

Out of these students, 620 (82.1%) were assigned to one of their first priority work groups, 103 students (13.6%) to their second or third priority level, and 32 students (4.2%) to a group of priority level four or more. Out of 127 teams of two or more students, 115 (90.6%) were assigned to one single work group, 8 (6.3%) were split up to two partial teams, 4 (3.1%) were split up to three or more parts. Only 14 students were assigned to a work group without any team mates.

The real-world instance shows a slightly different internal structure than the artificial instances of the same size: students comprising a team often arrange to enter the same priority values, which is not mapped in the generation of artificial instances. Many students also apply the same weights for the majority of their priorities.

Students in the UoC's programming class of 2013/14 were assigned to their work groups according to the generated solutions. After the inclusion of latecomers and some manual last minute changes, instructors and students were content with the assignments.

5 Conclusions and further research

This paper introduced the Team Priority Assignment Problem, a combinatorial optimization problem related to GAP, and showed that TPAP is NP-hard. To generate high-quality solutions in reasonable run time, the simulated annealing heuristic was adapted and applied to both randomly constructed instances and the real-world instance of the University of Cologne's 2013 programming class. The method yielded a solution where 82.1% of students were assigned to their

n	cost	group cost	team cost	run time	min. cost	max. cost	Avg. priority	Ratio of complete teams	σ	$\frac{\sigma}{avg.\ cost}$
100	24,668.0	19,903.0	4,765.0	89.3 s	24,516.3	25,131.2	1.61	36%	139.3	0.006
200	42,202.7	31,079.3	11,123.4	218.6 s	41,641.3	43,164.4	1.94	33.3%	384.8	0.009
300	59,200.8	46,828.4	12,372.4	511.5 s	57,895.6	60,241.7	2.85	55.1%	483.8	0.008
400	72,481.3	55,269.0	17,212.3	494.0 s	70,754.5	73,952.9	3.07	50.0%	790.6	0.011
500	81,514.9	59,698.4	21,816.4	627.5 s	80,025.9	83,238.6	3.38	48.9%	882.3	0.011
600	100,980.0	75,778.2	25,201.8	891.0 s	98,964.5	102,291.0	4.22	53.4%	888.7	0.009
700	112,042.1	80,826.7	31,215.4	990.2 s	110,348.9	114,052.0	4.30	44.6%	973.7	0.009
800	118,929.7	86,959.8	31,969.8	1,231.0 s	116,775.9	122,208.2	4.69	48.6%	1,160.3	0.010
900	134,983.8	95,319.6	39,664.2	1,473.7 s	131,857.6	138,456.9	5.03	45.2%	1,725.1	0.013
1000	147,726.9	104,604.6	43,122.3	1,689.4 s	144,416.8	152,688.7	5.48	54.9%	1,924.7	0.013
755	2.196,9	1.998,4	198,5	1.211,6	1.757,9	2.505,2	1,35	90,6%	175,4	0,080

Table 2: Results for random instances and UoC instance

first priority level work group, 13.6% to their second or third priority level, and 4.2% to a priority level of four or more. 90.6% of teams were assigned to one single work group, 9.4% were split up to two or more partial teams. The results show that the simulated annealing implementation of the described model generates high-quality solutions in reasonable run time which are feasible to apply to real-world instances.

Further steps might consist of the examination of different objective functions, because when applied to large teams, the power-of-four function does not show the desired behavior of discouraging the splitting off of single students. We will also consider different neighborhood relationships and extended sets of elementary modifications. To reliably reach better solutions, the exchange of whole or partial teams seems to be especially promising. Also, the formulation of a linear objective function and its embedding in a linear integer model will be considered.

6 Acknowledgments

This material is based in part upon work supported by the National Science Foundation under Grant Nos. I/UCRC IIP-1338922, AIR IIP-1237818, SBIR IIP-1330943, III-Large IIS-1213026, MRI CNS-0821345, MRI CNS-1126619, CREST HRD-0833093, I/UCRC IIP-0829576, MRI CNS-0959985, FRP IIP-1230661 and U.S. Department Transportation under a TIGER grant.

References

[1] Burkard, R., Dell'Amico, M., Martello, S.: Assignment Problems. Society of Industrial and Applied Mathematics, 2009.

[2] Catrysse, D.G., Van Wassenhove, L.N.: A survey of algorithms for the generalized assignment problem. In: European Journal of Operations Research 60, 1992, pp. 260-272.

[3] Díaz, J.A., Fernández, E.: A Tabu search heuristic for the generalized assignment problem. In: European Journal of Operations Research 132, 2001, pp. 22-38.

[4] Dréo, J., Pétrowski, A., Siarry, P., Taillard, E.: Metaheuristics for Hard Optimization. Springer, 2006.

[5] Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-completeness, Freemen Pub, 1979.

[6] Kuhn, H. W.: The Hungarian method for the assignment problem. In: Naval Research Logistics Quarterly Volume 2, Issue 1-2, 1955, pp. 83-97.

[7] Liang, T.T., Thompson, T.J.: A large-scale personnel assignment model for the navy. In: Decision Sciences, Vol. 18, 1987, pp. 234-249.

[8] Martello, S., Toth, P.: An algorithm for the generalized assignment problem. Operational research, 81, 1981, pp. 589-603.

[9] Martello, S., Toth, P.: Knapsack Problems. Chichester: John Wiley & Sons, 1990.

[10] Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., Teller, E.: Equation of state calculations by fast computing machines. In: Journal of Chemical Physics, Vol. 21, 1953, pp. 1087-1090.

[11] Munkres, J.: Algorithms for the Assignment and Transportation Problems. In: Journal of the Society for Industrial and Applied Mathematics, Vol. 5, No. 1, Mar. 1957, pp. 32-38.

[12] Osman, I.H.: Heuristics for the generalised assignment problem: simulated annealing and tabu search approaches. In: OR Spektrum 17, 1995, pp. 211-225.

[13] Ramalhino Lourenço, H., Serra, D.: Adaptive Search Heuristics for The Generalized Assignment Problem. In: Mathware & Soft Computing 7, 2000, pp. 1-15.

[14] Ross, G.T., Soland, R.M.: A branch and bound algorithm for the generalized assignment problem. In: Mathematical Programming 8, 1975, pp. 91-103.

[15] Savelsbergh, M.: A branch-and-prize algorithm for the generalized assignment problem. In: Operations Research, Vol. 45, No 6 (Nov. - Dec., 1997), pp. 831-841.

[16] Toroslu, I.H., Arslanoglu, Y.: Genetic algorithm for the personnel assignment problem with multiple objectives. In: Information Sciences 177, 2007, pp. 787-803.