SOME SPECIFIC HEURISTICS FOR SITUATION CLUSTERING PROBLEMS

Boris Melnikov, Alexey Radionov
Department of Mathematics and Information Science, Togliatti State Univ., Belorusskaya str., 14, Togliatti, 445667, Russia

Andrey Moseev, Elena Melnikova
Department of Telecommunication Software, Ulyanovsk State Univ., L.Tolstoy str., 42, Ulyanovsk, 432700, Russia

Keywords: Discrete optimization problems, Multi-heuristic approach, Clustering situations.

Abstract: The present work is a continuation of several preceding author's works dedicated to a specific multi-heuristic approach to discrete optimization problems. This paper considers those issues of this multi-heuristic approach which relate to the problems of clustering situations. In particular it considers the issues of the author's approach to the problems and the description of specific heuristics for the problems. We give the description of a particular example from the group of "Hierarchical Clustering Algorithms", which we use for clustering situations. We also give descriptions of some common methods and algorithms related to such clustering. There are two examples of metrics on situations sets for two different problems; one of the problems is a classical discrete optimization problem and the other one is a game-playing programming problem.

1 INTRODUCTION AND PRELIMINARIES

This paper is a continuation of several preceding authors' works dedicated to a specific multi-heuristic approach to discrete optimization problems (DOPs). We should mark that the given list of literature consists almost only of the works of the present paper's authors. First of all it is caused by the fact that the authors did not meet publications about application in discrete optimization problems of all the heuristics *simultaneously*. Among those works we would mark separately the following:

- description of the approach to DOPs in general (Melnikov, 2005; 2006);
- description of specific heuristics for particular DOPs (Belozyorova and Melnikov, 2005; Melnikova and Radionov, 2005; Melnikov, Radionov, and Gumayunov, 2006);
- description of heuristics for game-playing programs (Melnikov and Radionov, 1998; Melnikov, 2001; Melnikov, Radionov, Moseev, and Melnikova, 2006).

However this division is mostly conventional because each of those papers contains partitions and heuristics concerning any of the three areas mentioned. The object of each of DOPs is programming anytime algorithms, i.e., the algorithms, which can provide a near-to-optimal solution in real time.

The considered DOPs will be briefly (and two of them quite properly) described in this paper. Here in the Introduction we will very briefly describe the approaches or actually the heuristics of the mentioned multi-heuristic approach; the text is given according to (Melnikov, 2005). So, the methods of solution of DOPs are constructed on the basis of special combination of some heuristics, which belong to different areas of the theory of artificial intelligence. Firstly, we use some modifications of truncated branch-and-bound methods (TB&B); moreover in this paper we will frequently refer to TB&B. Secondly, for the selecting immediate step of TB&B using some heuristics, we use dynamic risk functions. Thirdly, simultaneously for the selection of coefficients of the averaging-out, we also use genetic algorithms. Fourthly, the reductive self-learning by the same genetic methods is also used for the start of TB&B.

In this paper we consider those issues of the multi-heuristic approach to DOPs which are connected with the problems of clustering (or classifying; here we do not consider the difference between these two terms for in the considered problems they have the same meaning) situations, actually with the author's approach to those problems, with description of the heuristics specific for those problems, and with applying the corresponding methods. Let us mention two works which became classical. The first one is about usage of clustering algorithms in artificial intelligence problems, see in (Lifschitz, 1990). The second one is a classical work on solving problems "by analogy" (in author's opinion this subject is quite popular nowadays), see in (Carbonell, 1983)..

Section 2 of this paper contains the description of an example from the group of "Hierarchical Clustering Algorithms"; the authors find this variant of algorithm the most efficient for the considered problems of situations clustering in application to AI problems and some empirical confirmations of this fact are given below. The authors apply this algorithm to problems in other areas (for more detail see Section 2).

In any case a problem of clustering rather often demands development of a specific metric. Moreover when we fix a concrete algorithm of clustering (independent of any particular metric) we can say that "if there is a metric then there is a clustering" and we do not simplify the situation too much. In our case we need to develop metrics on situation sets and two examples of that are considered in Sections 3 and 4 of this paper. One of the problems is a classical discrete optimization problem and the other one is a game-playing programming problem.

Section 5 contains descriptions of some common methods and algorithms related to clustering. We apply these algorithms to different DOPs, both classical and intellectual game-playing programming.

Some practical results of usage of our programs in two descrete optimization problems (minimization of non-deterministic finite automata and DNF) are mentioned in conclusion (Section 6). The programs contain implementations of all the heuristics described in this paper and in the previous publications of the authors. We can note in advance that the results of our programs execution are purely comparable to the best programs of other developers (found in the Internet and others).

2 AN EXAMPLE

The following Section describes an example from the group "Hierarchical Clustering Algorithms", applied by the authors in a number of DOPs. Let us give a mathematical description of this algorithm. We should note in advance, that here appear a lot of problems, connected with building the most efficient variants to carry out this very variant – however, we are not going to study these problems in the present article.

So, let R_{ij} be the minimal distance between the elements of clusters numbered i and j, and r_i be the maximal distance between the elements of cluster i. Common algorithms of clustering and algorithms of its quality evaluation generally use the values

$$f(\min_{i,j}(R_{ij})) + g(\max_{i}(r_i)),$$

where f and g are specific increasing functions (chosen depending on a particular problem, particular algorithm and so) and i and j take on all possible values. First of all we mean different variants of algorithms from the group of "Hierarchical Clustering Algorithms". For example see the site `http://www.elet.polimi.it/upload/matteucc/Clustering/` and also the information that can be found by links from this site.

As distinct from common algorithms the authors use the same formula but with other meanings of r_i values. In particular let:

- i be the considered cluster,
- $m_1,...,m_n$ be all its elements,
- l_{mn} be the distance between its elements m and n,
- $(M_1,..., M_N)$ be some finite sequence consisting of the cluster's elements $m_1,...,m_n$ and including each of them at least once.

Then we consider

$$r_i = \min_{(M_1,...,M_N)} \left(\max_{0<K<N} (l_{M_K M_{K+1}}) \right).$$

Note that although the given definition is not an algorithm the simplest possible algorithms of calculating r_i values built on its base are not difficult. There are some differences between implementation of these algorithms and the common ones (i.e., implementation of the algorithms which can be found by the links on the site mentioned before) but we will not concentrate on those differences. (Actually those differences could be the subject of a separate publication, which practically would not be related to the present paper. One of those differences is concluded in the following: we step by step increase the criteria of combining two clusters into one, i.e., lowering the clustering threshold with help of a specific auxiliary algorithm which uses precomputed distances defined on the current clusters.) As it was mentioned above, there can be found some interesting problems focused on efficient realization of this algorithm variant. However we will not study these problems in the article either. Instead of considering such algorithmic problems we are going to approve empirically the expediency of using this algorithm in the considered DOPs.

Thus we have shown a formal description of one specific clustering algorithm. As it was mentioned in Introduction the authors suppose that one of its applications could be the problem of forming the initial screen when searching in the Internet. In (Melnikova and Radionov, 2005), two different possible variants of using this algorithm in such a problem are given and a preliminary version of search engine was developed basing on this approach. However here we are more concerned about another possible application area of the algorithm and this area is discrete optimization problems.

The following question may arise: why do we need this very algorithm, what makes it better than the other ones described above? (Another question may arise why in our case we need to solve problems "by analogy" though the goal of each anytime-algorithm is in achieving one pseudo-optimal solution. A possible answer to this question is given in Section 5 of this paper.) It is impossible to give the exact answer, as it always seems in case of heuristics. But practical programming and the solution of several DOPs has shown that this algorithm is the best to illustrate possible locations of situations. Here we mean the situations both for a certain DOP and for game-playing programs. We attempt to focus on situations not only within one cluster, but also in an order which corresponds to the possible sequence (M_1,…M_n) in the formulae above and what is more important to the selection of the situations which are included more than once in into this sequence (let us call them key situations). Situations are sorted in the given way for ordered processing of the situations, for solving the corresponding subtasks in a similar order, and we certainly start with the key situations. (We mark in advance that our vision of the term subtask will bee defined more exactly in the next section.) By such situations sorting we achieve successful solving of problems "by analogy" because the situations which are close in a "good" metric most likely have similar solutions. Here we first have in mind the possibility to make the same step in TB&B. And not just possibility to make the same step, but heuristic choice and receiving the same separating element for such step in two subtasks close in the metric.

So the authors hope that they have given the answer to the possible question about necessity of using the described algorithm of clustering in different DOPs. And as it was mentioned above to do clustering with help of this algorithm we need to find a metric on the situations set (subtasks set). The following two Sections of the paper are dedicated to description of such metrics for two different DOPs.

3 METRICS FOR SITUATIONS IN ONE PROBLEM

In this section we give a metric for the problem of vertex minimization in nondeterministic finite automata of Rabin-Scott. However the authors leave this title of the section because in principle similar heuristics can be applied to quite different discrete optimization problems.

Let us explain the task setting (for more details see (Melnikov, 2000) and others). Some matrix is filled with numbers 0 and 1 (in terms of (Melnikov, 2000), numbers 1 correspond to elements of binary relation #). Let us use the term Grid for any couple of subsets of columns and rows of the matrix if those subsets meet the following two conditions:

- at the intersection of each column with each row of those subsets 1 is placed;
- we can not add any row or column to those subsets without breaking the first condition.

In the set of all possible grids we need to find a subset which includes all 1s of the given matrix. And to reach the requirement of vertex minimization in nondeterministic finite automata we should find that one of the subsets which contains the minimal number of grids in it. (One description of the sufficient requirement is given in (Polák, 2004). Besides in present time the authors of the paper are preparing a publication with a description of such algorithm based on description of the set of all possible arcs of an automation, which is given in (Melnikov, and Sciarini-Guryanova, 2002). To be more specific it is based on presence of analogues of all these arcs in the given automation. However here we consider only the requirement of vertex minimization so there is no need to consider arcs and each grid is a special description of an automation state.)

We can demonstrate that the problem is not trivial by showing the following simple example. As usual for such examples it is specially designed to make greedy algorithms inefficient. So, let the initial matrix be set by the Fig.1.

$$\begin{array}{c|cccccc|}
 & U & V & W & X & Y & Z \\
\hline
A & 1 & 1 & 1 & 1 & 0 & 0 \\
B & 1 & 1 & 1 & 0 & 1 & 0 \\
C & 1 & 1 & 1 & 0 & 0 & 1 \\
D & 1 & 0 & 0 & 0 & 0 & 0 \\
E & 0 & 1 & 0 & 0 & 0 & 0 \\
F & 0 & 0 & 1 & 0 & 0 & 0 \\
\hline
\end{array}$$

Figure 1: Initial Matrix.

Then the optimal solution does not contain the maximal grid, i.e. the grid of columns U, V, and W and rows A, B and C. Optimal solution consists of the tree single-row grids (the same A, B and C but now each of them contains all available 1) together with three similar single-column grids. Obviously real tasks are much more complex but as for smaller tasks this one is quite interesting.

Now we are going to consider some different metrics but first we need to explain some general notations. Let X and Y be some sets, $n=|X\cap Y|$ be the number of elements in their intersection, $N=|XUY|$ be the number of elements in their union. Then we will write $\Omega(X,Y)=1–n/N$.

Let us consider the heuristic for definition of a metric on the grids set. (We suppose that any possible metric should deliver value of 0 if the grids are totally the same and value of 1 if the grids have no common elements, i.e. common matrix cells filled with 1.) As a result of practical programming including testing of several alternatives we have found that the following heuristic is the most efficient for the problem. Let X_1 and X_2 be sets of the numbers of the grids' rows 1 and 2, Y_1 and Y_2 be the same sets for the columns. Then we will use the following value as a metric

$$(\alpha\cdot\Omega(X_1,X_2)+ (1- \alpha)\cdot\Omega(Y_1,Y_2)),$$

where α is some coefficient to be adjusted with help of genetic algorithms. (Actually it also depends on the number of elements in the mentioned arrays, but we do not discuss genetic algorithms in this paper.) Obviously if the grids are the same then we get 0. Though we do not always get 1 if the grids have no common cells with value of 1, nevertheless the given metric is quite efficient for the considered problem.

Now let us consider possible heuristics for definition of a metric on the subtasks set. To do this we need to define more exactly what is subtask (though we will not give the complete definition, it is just impossible) and first let us show fragments of the corresponding programs in C++, containing descriptions of task and subtask. Here we reach two goals at the same time: firstly we show a possible variant of implementation of the approach to DOPs programming and secondly we introduce all the auxiliary elements needed for our definition of subtask. We hope that the comments placed in the listing will help to read the source code.

First of all here is the class describing the whole DOP.

```
class MyTask : public MAG {
  // the ancestor contains
  // the initial matrix
  // and all yet generated grids
private:
  AST sbtAll;
    // all possible generated subtasks
  MySubTask* pstAnyTime;
    // the best of yet solved tasks;
    // it is NOT included in sbtAll
    // ...
};
```

Additional comments. We use the following auxiliary classes:

- ancestor class `MAG` (abbreviation for Matrix, Array, Grid) includes data of the initial matrix and of the grids which are generated to the moment;
- class `AST` ("array of subtasks") is an array of subtasks; when using MFC library this class should be derivate of the class `CPtrArray` which includes only pointers to defined below subtask objects (of the class `MySubTask`).

Then here is the class describing a subtask; we are listing the variant using MFC.

```
class MySubTask {
private:
  MyMatrix* pMatrixThis;
    // already filled HERE matrix
  double rQuality;
  MyWordArray* pArrYes;
  MyWordArray* pArrNo;
    // both arrays contain
    // numbers of grids
    // ...
};
```

Additional comments. The matrix `*pMatrixThis` is of the same type as the matrix included in the class MAG; obviously at the start of TB&B this matrix is filled with 0 only, and when receiving any current solution it coincide with the matrix included in the class MAG. The value `rQuality` is a boundary which is used in TB&B execution. The array `*pArrYes` contains the numbers of the grids which form the matrix `*pMatrixThis`. According to the common principles of TB&B a subtask must include the list of the grids which cannot be included in forthcoming solution and `*pArrNo` contains this list. References to the grids are made by their numbers included in the class MAG. Thus there is no direct access to the grids from the class `MySubTask` but the logic of this approach to the problem i.e. the logic of TB&B application does not demand such access.

Basing on the introduced above metric on grid we define a metric on subtasks. To do this let us first define a metric for any two arrays of grids. Let $X=\{x_1,\ldots,x_m\}$ be the first array, $Y=\{y_1,\ldots,y_n\}$ be the second one. Then the metric itself is

$$\rho(X,Y) = (\textstyle\sum_{i,j} \Omega(x_i, y_j)) / (m\cdot n),$$

where i and j take on all possible values. Then let X_1 and Y_1 be arrays *pArrYes and *pArrNo for the first subtask, X_2 and Y_2 be the corresponding arrays for the second one. Let us denote

$$A = \rho(X_1,X_2),\ B_1 = \rho(X_1,Y_2),\ B_2 = \rho(X_2,Y_1),$$

and as the distance between the subtasks we will use the value

$$(1+2\cdot\alpha)\cdot A - \alpha\cdot(B_1+B_2),$$

where α is again some coefficient to be adjusted with help of genetic algorithms. (Actually the last formula is true only for "close" subtasks. In the other case, for example in case of high values of the sum B_1+B_2, we use other formulae, however as before we will not get deeply into details.)

When we use this metric in solving subtasks "by analogy" as it was said above we make efficient versions of programs because according to the metric's description we have similar sets of grids in the arrays *pArrYes and *pArrNo for close in the metric subtasks. However this metric demands precomputing of distances between any two of the grids generated to the moment. That is why the authors often use another metric which is somehow analogous to the metric given above for the grids.

So let us describe another variant of metric on a subtasks set. Let P_1 be the set of cells of the first subtask's matrix with value of 1; P_2 be the same for the second subtask. Then as a metric we use the value $\Omega(P_1,P_2)$. Note that the formulas here are mush simpler than in the previous variant but if computing of the distances on the set of grids is made beforehand then the calculations for the second variant of metric take quite longer. These calculations bay be demanded as auxiliary for different subtasks of TB&B and in some variants of TB&B are actually made beforehand. Besides see the brief description of possible algorithms managing such calculations in Section 5.

4 METRICS FOR SITUATIONS IN GAME PROGRAMS

This section describes one problem of nondeterministic game-playing programming and metrics used for clustering of situations in this game. However we (as in the previous section) leave the general title because the offered approach can be used in other problems of non-deterministic game-playing programming.

The particular nondeterministic game in which we use situations clustering is called "Omega" game; a similar game was described in one of Martin Gardner's works. Let us briefly describe the rules of this game.

In the game a square matrix (with N rows and N columns) is used. Two players take turns to fill the matrix with natural numbers. After filling the whole matrix (both players make N^2 moves in aggregate) the first player calculates the sum of the numbers productions by rows, the second player calculates the sum of the productions by columns. The difference between the sums is the first player's profit and the second player's loss. The numbers which the players use to fill the matrix are chosen at random from the range between 1 and N^2 before each move.

For this game even at the dimension of 3 (3 rows and 3 columns) there is no chance to find the optimal move with help of exhaustive search. That is why here we have to use TB&B modified in a specific way according to nondeterministic nature of the game (for details see (Melnikov, Radionov, Moseev, and Melnikova, 2006)). In particular we use the method of risk functions developed by the authors for programming of nondeterministic games. Besides, some specific heuristics are used in order to shorten the search.

In most cases at a regular step of TB&B we need to classify the situation to choose one of the heuristics or to define the particular values of the used heuristics' parameters for instance the values of the risk functions' parameters. Moreover, an efficient method of situations classifying often helps to determine such an order of the moves processing which can shorten the search.

Before we can define a metric on the set of situations of this game we need to define the term situation itself (or subtask in term of the previous section). A situation in this case is a game matrix partly filled with numbers i.e. a state of the game matrix after a move of one of the players. Thus we are going to define a metric on the set of partly filled matrixes.

To find the distance between two partly filled matrixes let us fill their empty cells with average of distribution of the random value. In our case we should put the value $(N^2+1)/2$ into all empty cells of both matrixes. Now for each matrix we find the products of numbers by columns and by rows. Then we permute the rows and columns of each matrix in ascending order of the corresponding products. Note that the result of this game does not change after permutation of rows or columns of its matrix. Let S be the sum of absolute values of the differences between the corresponding products of the matrixes (2*N differences in aggregate). The value of S can be considered the distance between the matrixes. We can also use a normalized variant i.e.

$$S/(2*N*(N^{2*N}-1)).$$

As is easy to see both variants are equal to 0 when the matrixes are the same. They also equal to 0 when the matrixes are equal to permutation and this property corresponds to the game specific. Besides, the normalized variant can never exceed 1 and it can reach this value only in the case when one matrix is completely filled with 1s and the other one is completely filled with N^2.

The given metric can be used in the described above approach of situations clustering.

5 SOME COMMON ALGORITHMS

In one of the previous sections the following question was asked: why in our case we need to solve problems “by analogy” though the goal of each anytime-algorithm is in achieving one pseudo-optimal solution. We can give the following answers. First, we do not know which one of the considered subtasks will deliver the pseudo-optimal solution (which will replace the current one i.e. *pstAnyTime according to the given program notation). Second, according to one of the main principles of the branch-and-bound method (not only in the truncated one) two different subtasks can not deliver the same final solution. Therefore the current pseudo-optimal solutions for different subtasks will certainly be different. However as is easy to see the given answers do not cancel the possibility (and maybe necessity) of solving similar (i.e. close in metric) tasks by analogy.

Thus solving problems by analogy is one of the common approaches which the authors use for different DOPs. Among the other common approaches we should firstly mark the following one.

As we said in Section 3 sometimes it is good to have a metric on a set of grids (or generally on a set of auxiliary objects common for several subtasks). However it often (including the described problem) takes too much time to build such metric on set of all situations. In this connection there can be used the following algorithm. In our implementations of TB&B we use such auxiliary algorithms one by one: finding the new metric value for a pair of auxiliary objects of the subset (grids and so);

- adjusting the precomputed metric value for a pair of subtasks;
- selection of the next subtask from the list if subtasks demanding solution;
- selection of an auxiliary object (grid or so) from the corresponding set; this object will be used as the separation element for the next step of TB&B;
- building the left and the right subtasks basing on the selected separation element and including these subtasks in the corresponding data structure for subsequent solution.

Thus we can say that here we have briefly described TB&B with application of algorithms connected with situation clustering. Remark also, that:

- The authors have just started to use this auxiliary algorithms in practical programming and its efficiency is still under question. Besides we should mark that the other possible solution of the problem is implementation of distributed calculations. However it is absolutely separate problem and the authors did not yet start to solve it in practice.
- In Section 3 we did not consider such algorithms of precomputing. For the problem of minimization of nondeterministic finite automata the formulae for the metric precomputing are similar to the ones given in Section 3. However they use only those pairs of grids which have their metrics already computed.
- Actually the most efficient data structure for keeping the subtasks sets is a specific variant of correctly filled tree (or heap in other terms; see (Cormen, Leiserson, Rivest, and Stein, 2001) and others).
- At the same time we build the so called sequence of the right problems (SRP). In (Melnikov,

2005), there is information about this auxiliary algorithm but it is quite important so we are giving a part of its description from that paper. Each time, when we obtain the next right problem (let us call it problem T) we make at the same time also the SRP, i.e., T, then the right problem of T, then the right problem of the right problem of T, etc. Certainly, we make each time also the corresponding left problems, i.e., the left problem of T, then the left problem of the right problem of T, etc. This process finishes:

– when we obtain a trivial problem (e.g., of dimension 1), then we use its solution (i.e., its bound, and also the obtained path, and similar behavior) by the *current* quasi-optimal solution of the considered anytime algorithm;
– or when we obtain the big value of the bound, for example, if this value is more than the current (existing) quasi-optimal solution.

In practice, such process of SRP-constructing does not require a lot of time, and the increasing the dimension of the list of problems for the solution in the future is very reasonable.

6 CONCLUSION. SOME PRACTICAL RESULTS

So in this paper we considered some issues of the offered by the authors multi-heuristic approach to discrete optimization problems, the issues connected with problems of situations clustering. In conclusion we will mark the following circumstances.

In auxiliary algorithms we often use self learning with help of genetic algorithms and not only in cases evidently marked in Section 3. Generally you can find more information about the author's usage of genetic algorithms in the mentioned above references (Melnikov and Radionov, 1998; Melnikov, 2005; Belozyorova and Melnikov, 2005; Melnikova and Radionov, 2005; Melnikov, Radionov, Moseev, and Melnikova, 2006; Melnikov, Radionov, and Gumayunov, 2006), mainly in (Melnikov, 2001; 2006).

We also should mark once more, that the given list of literature consists almost only of the works of the authors of the present paper. However this is a consequence of the fact that we offer our own approach to solving DOPs generally.

As it was mentioned above, the results of our programs execution are about the same as the ones of the programs found in the Internet. The best of available links can be found at `http://citeseer.ifi.unizh.ch/fiser02set.html` or at `http://www.dei.isep.ipp.pt/~acc/bfunc/.` Now we are going to show one slightly different approach to describing the practical results, see below.

While testing, we set the time for our anytime algorithm (see the tables). We also set the dimension of the problem – i.e., the number of rows for NFA (the number of columns depends of the last value also by special variate) and the number of variables for DNF – not the numbers of grids for NFA and planes for DNF, the last values are also special variates depending on previous ones.

The clock speed of the computer was about 2.0 GHz. If we choose the time under 10 minutes, we make the averaging-out about 50 solutions.

And the values of cells have the following meaning. For each cell, we made corresponding tests. For each test, we set the number of grids/planes for the given problem (certainly, we did not use this information in the program) and obtain the value of grids/planes found by anytime algorithm. Then we counted comparative improvement of this value (+) or the worsening (–). The possibility of positive values is the corollary of the fact, that, e.g. for the DNF, two planes of dimension k could form one plane of dimension k+1. The values were averaged; they are written in the table in percents. (I.e., +0.22 means that the mean value is better than the a priori given than 0.22%.)

Thus, below are the practical results.

Table 1: Nondeterministic Finite Automata.

NFA	20–23	40–45	60–65
01 sec	1.76	–0.58	–0.02
10 sec	–0.55	–0.17	+0.22
01 min	–0.03	+0.45	+1.00
10 min	0	+1.06	+1.07
01 h	0	+1.07	+1.17

Table 2: Disjunctive Normal Forms.

DNF	20–22	25–27	30–31
01 sec	–8.5	–2.2	–1.9
10 sec	–1.21	–0.70	–0.43
01 min	–0.73	–0.65	–0.43
10 min	–0.03	–0.01	–0.01
01 h	–0.03	–0.01	0

The obtained results are near to 100%; this fact shows that the approach proposed in this paper could be applied in the future. And in the next papers, we are going to give the practical results for two other problems mentioned in Section 2.

The authors hope to continue the series of papers about applying the multi-heuristic approach to solution of different DOPs. In this connection we should note that some subjects of the new algorithms, programs and publications were briefly mentioned in the present paper.

REFERENCES

Belozyorova, A., and Melnikov, B., 2005. Using Heuristics Complex for the Problem of Making Diagram of Nuclide Transformations, In *2nd Scientific Conference "Tools and Techinques of Information Processing"*, Moscow State Lomonosov Univ. Ed. (2005) 208–210 (in Russian).

Carbonell, J., 1983. Derivational Analogy and Its Role in Problem Solving, *AAAI* (1983) 64–69.

Cormen, T., Leiserson, C., Rivest, R., and Stein, C., 2001. *Introduction to Algorithms*. – MIT Press and McGraw-Hill, 2001.

Lifschitz, V., 1990. Frames in the Space of Situations, *Artificial Intelligence* 46 (1990) 365–376.

Melnikov, B., and Radionov, A., 1998. On the Decision of Strategy in Nondeterministic Antagonistic Games", In *Programming and Computer Software (Programmirovanie, Russian Acad. Sci. Ed., English translation)*, Vo.24, No.5 (1998).

Melnikov, B., 2000. Once More About the State-Minimization of the Nondeterministic Finite Automata, *The Korean Journal of Computational and Applied Mathematics*, Vo.7, No.3. (2000) 655–662.

Melnikov, B., 2001. Heuristics in Programming Nondeterministic Games, In *Programming and Computer Software (Programmirovanie, Russian Acad. Sci. Ed., English translation)*, Vo.27, No.5 (2001) 277–288.

Melnikov, B., and Sciarini-Guryanova, N., 2002. Possible Edges of a Finite Automaton Defining a Given Regular Language, *The Korean Journal of Computational and Applied Mathematics*, Vo.9, No.2 (2002) 475–485.

Melnikov, B., 2005. Discrete Optimization Problems – Some New Heuristic Approaches, In *8th International Conference on High Performance Computing and Grid in Asia Pacific Region*, IEEE Computer Society Press Ed. (2005) 73–80.

Melnikova, E., and Radionov, A., 2005. A Construction of Taxonomic Hierarchy of Documents Using Risk Functions, In *2nd Scientific Conference "Tools and Techinques of Information Processing"*, Moscow State Lomonosov Univ. Ed. (2005) 103–106 (in Russian).

Melnikov, B., 2006. A Multi-Heuristic Approach to the Problems of Discrete Optimization, In *Cybernetics and Systems Analysis* (Ukranian Acad. Sci. Ed.), 2006, No. 5, accepted (in Russian).

Melnikov, B., Radionov, A., and Gumayunov, V., 2006. Some Special Heuristics for Discrete Optimization Problems, In *8th International Conference on Enterprise Information Systems*, Cyprus (2006) 91–95.

Melnikov, B., Radionov, A., Moseev, A., and Melnikova, E., 2006. Heuristics for Working With Searching Tree in Non-Deterministic Games, *ICGA Journal*, 2006, submitted.

Polák, L., 2004. Minimization of NFA Using the Universal Automaton, *CIAA* (2004) 325–326.