Solving the MDBCS Problem Using the Metaheuric– Genetic Algorithm

Genetic algorithm for the MDBCS problem

Milena Bogdanović
University of Niš
Teacher Training Faculty
Partizanska 14, Vranje, Serbia

***Abstract*— The problems degree-limited graph of nodes considering the weight of the vertex or weight of the edges, with the aim to find the optimal weighted graph in terms of certain restrictions on the degree of the vertices in the subgraph. This class of combinatorial problems was extensively studied because of the implementation and application in network design, connection of networks and routing algorithms. It is likely that solution of MDBCS problem will find its place and application in these areas. The paper is given an ILP model to solve the problem MDBCS, as well as the genetic algorithm, which calculates a good enough solution for the input graph with a greater number of nodes. An important feature of the heuristic algorithms is that can approximate, but still good enough to solve the problems of exponential complexity. However, it should solve the problem heuristic algorithms may not lead to a satisfactory solution, and that for some of the problems, heuristic algorithms give relatively poor results. This is particularly true of problems for which no exact polynomial algorithm complexity. Also, heuristic algorithms are not the same, because some parts of heuristic algorithms differ depending on the situation and problems in which they are used. These parts are usually the objective function (transformation), and their definition significantly affects the efficiency of the algorithm.**
By mode of action, genetic algorithms are among the methods directed random search space solutions are looking for a global optimum.

Keywords- graph theory; NP-complete problems; the degree-bounded graphs; Integer linear programming; genetic algorithms.

I. INTRODUCTION

General problems of degree-constraind graphs, consider the nodes of weight or weight on the vertices, where the goal is to find the optimal weighted graph, with set limits for levels of subgraph nodes. This class of combinatorial problems has been extensively studied for use in designing networks. If the input graph is bipartite, ie, if the set of its nodes can be broken down into two nonempty disjoint subsets so that each vertices has one end in each of the two subsets, then these problems are equivalent to the classical transportation problem in terms of operational research. The reason for this extensive study and research issues listed above, lies in their wide application in the areas of networks and routing algorithms.

In the Theory of complexity, NP (nondeterministic polynomial time) is a set of decision problems that can be solved by nondeterministic Turing machine. The importance of this class of decision problems is that it contains many interesting problems of search and optimization, where we want to determine whether there is some solution to the problem, but whether this is the optimal solution. Therefore, the challenge with NP problem is to find the answer in an efficient manner, as an effective way to verify the response, ie. solution already exists. Since many important problems in this class, intensive efforts were invested to find in polynomial time algorithms for solving problems in class NP. However, a large number of NP problems has resisted these efforts, and apparently, they require time polynomial is not even close! Are these problems really are not solvable in polynomial time is one of the biggest open questions in computer science.

The easiest way to prove that an problem a NP-complete problem is to first prove that the NP and then to an already known NP-complete problem down to him. It is therefore useful to know the various NP-complete problems.

The problems of class NP-complete are classified into the following groups:

1) *Covering and Partitioning*
2) *Subgraphs and Supergraphs*
3) *Vertex Ordering*
4) *Iso- and Other Morphisms*
5) *Miscellaneous*

From the group Subgraph and supergrafs distinguishes the following NP-problems:

1) *MAXIMUM INDEPENDENT SET*
2) *MAXIMUM INDEPENDENT SEQUENCE*
3) *MAXIMUM INDUCED SUBGRAPH WITH PROPERTY* P
4) *MINIMUM VERTEX DELETION TO OBTAIN SUBGRAPH WITH PROPERTY* P
5) *MINIMUM EDGE DELETION TO OBTAIN SUBGRAPH WITH PROPERTY* P
6) *MAXIMUM INDUCED CONNECTED SUBGRAPH WITH PROPERTY* P
7) *MINIMUM VERTEX DELETION TO OBTAIN CONNECTED SUBGRAPH WITH PROPERTY* P

8) MAXIMUM DEGREE-BOUNDED CONNECTED SUBGRAPH
9) MAXIMUM PLANAR SUBGRAPH
10) MINIMUM EDGE DELETION K-*PARTITION*
11) MAXIMUM K-*COLORABLE SUBGRAPH*
12) MAXIMUM SUBFOREST
13) MAXIMUM EDGE SUBGRAPH
14) MINIMUM EDGE K-*SPANNER*
15) MAXIMUM K-*COLORABLE INDUCED SUBGRAPH*
16) MINIMUM EQUIVALENT DIGRAPH
17) MINIMUM INTERVAL GRAPH COMPLETION
18) MINIMUM CHORDAL GRAPH COMPLETION

The problem MDBCS is stated as follows:

INPUT: a graph $G = (V, E)$, the function of weight on the vertices (or weighting function) $w{:}E \rightarrow \mathbb{R}^+$, and an integer $d \geq 2$.

SOLUTION: a subset of $E' \subseteq E$ such that the subgraph $G' = (V, E')$ is connected and that there is no node with degree exceeding d.

MEASUREMENTS: Total weight of found subgraph, ie. $\sum_{e \in E'} w(E)$.

Considering an undirected graph $G = (V, E)$, where E is the set of vertices, and V is a set of nodes, and let $|V| = n, |E| = m$. Let $G = (V, E)$ is connected (but not necessarily) a graph with a weighting function: $w{:}\, E \longmapsto \mathbb{R}^+$. For simplicity, we can only say that the graph G w-weighted graph. For an arbitrary subset $E' \subseteq E$, with G'denotes subgraph of G induced by E', and $w(E')$ denote the sum of weight on the vertices of E', ie, $w(E') = \sum_{e \in E'} w(e)$.

Let $G = (V, E)$ is an undirected graph, let $d \geq 2$ an integer, and let $w{:}\, E \longmapsto \mathbb{R}^+$weighting function. The problem limits the maximum degree of connected subgraph (MDBCS) consists in finding a subgraph $G' = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$, so that the subgraph G' is connected, and $\sum_{e \in E'} w(e)$ has a maximum value.

The constants in the ILP model are: E - denote arrays of sets, and values of functions w in them.

Variables for the ILP model are:

$$x_e = \begin{cases} 1, e \in E' \\ 0, e \notin E' \end{cases},$$
$$y_e = \begin{cases} 1, e \in T' \\ 0, e \notin T' \end{cases},$$

where $e \in E$, and T' is a spanning tree for the subgraph G',

$$z_i = \begin{cases} 1, i \in V' \\ 0, i \notin V' \end{cases}.$$

ILP formulation of the model for finding the maximum degree of a connected subgraph constraints is given below, part of the paper [4].

Determine

$$max \sum_{e \in E} w_e x_e \quad (1)$$

with conditions:

$$\sum_{e \ni i} x_e \leq d, \ \forall i \in V, \quad (2)$$

$$y_e \leq x_e, \ \forall e \in E, \quad (3)$$

$$\sum_{e \in E} y_e = -1 + \sum_{i \in V} z_i, \quad (4)$$

$$x_e \leq z_{i_e}, \ \forall e \in E, \quad (5)$$

$$x_e \leq z_{j_e}, \ \forall e \in E, \quad (6)$$

$$\sum_{i_e, j_e \in S} y_e \leq |S| - 1, \ \forall S \subseteq V, \ |S| \geq 3. \quad (7)$$

Graph G is not oriented. In the ILP model, the formula (5), indicates the starting node i_e of the vertice e, while in formula (6), j_e indicates an incoming (final) node of the vertice e.

The objective function, given by (1), maximizes the sum of the total weight.

Condition (2) ensures that the subgraph G' each node has the most d vertices leading from the G'.

The condition given by (3), ensures that the subgraph G' is a superset of T' and if T' as a spanning tree is connected, it follows that the subgraph G' also connected.

Conditions (4), (5) and (6), provided that the candidate for the spanning tree T' has as many nodes in the subgraph G' a vertices, minus 1. Finally, condition (7) guarantees that the candidate for the spanning tree T' has no cycle. Therefore, the conditions (4), (5), (6) and (7) together ensure that T' is the spanning tree for the subgraph G'.

The following theorem proves the correctness of the above ILP model.

Theorem 1. [4] The MDBCS problem can be solved if and only if the following conditions (2) - (7) holds or their equivalent set of conditions.

II. METAHEURISTIC GENETIC ALGORITHM FOR SOLVING THE MDBCS PROBLEM

Genetic algorithms (GA) are a family of algorithms, which use some of the genetic principles that are present in nature, in order to solve certain computational problems. These natural principles are: inheritance, crossover, mutation, survival of the best custom (survival of the fittest), migration and so on. These algorithms can be used for solving various classes of problems because they are fairly general nature. In this case, they are used in the optimization problem - finding the optimal parameters of a system ([5]).

In a narrow sense, the notion of a genetic algorithm applies only to the model introduced by John Holland in his book „*Adaption in natural and artificial systems*“, 1975. ([10]). Holland is considered the creator of this metaheuristic and basic settings of his earliest works are valid even today. In a broader sense, genetic algorithm is any algorithm that is based on a population and operators of selection, crossover and mutation, which are used to obtain new points in the search space.

Genetic algorithm is applied to the final set of individuals called the *population*. Each individual in the population is represented by a series of characters (genetic code) and corresponds to a solution in search space. Coding can be binary

or a transliteration of higher cardinality. Encoding solutions is an important step of genetic algorithm because, inadequate choice of code can lead to poor results regardless of the rest of the structure of the algorithm.

The diversity of genetic material is provided by generating the initial population randomly. It can be used and some heuristics to generate initial population, or part thereof, of course, if the heuristics perform relatively quickly and significantly if not reduce the diversity of genetic material.

Individuals assigned to each function adaptation (fitness function) that evaluates the quality of given individuals, as well as individual solutions in the search space. The task of genetic algorithms is to provide a constant, from generation to generation, improving the adaptability of the absolute population. This is realized by applying successive genetic operators selection, crossover and mutation, thus gaining a better solution given a particular problem.

Mechanism of selection favoring above average fitted individuals and their above-average custom parts (genes), which receive a higher chance of their own reproduction in the formation of a new generation. In this way, less fitted individuals and genes get less chances to play, and gradually dying out. Contribution to diversity of genetic material from your operator crosses that controls recombination genes of individuals. As a result of the crossing structure is obtained, although non-deterministic, the exchange of genetic material between individuals, with the possibility that well-adjusted individuals generate better individuals. The mechanism of crossing operators and relatively less fitted individuals, with some well-adapted genes, gets his chance to recombination of good genes produce well-adjusted individuals. However, using multiple selection and crossbreeding can result in loss of genetic material, ie, some regions of space results become available. The operator performs a random mutation of a particular gene, given the small probability p_{mut}, which can restore the lost genetic material in the population. This is the basic mechanism for preventing premature convergence of genetic algorithm to a local extreme, (see [3]).

Operators are applied until a stopping criterion is met, for example. reached the maximum number of generations, the same quality of solutions in a number of generations, finding the optimal solution, the best individual was repeated a maximum number of times, limited the time of executing the genetic algorithm, the termination by the user and so on.

As the most important aspects of a genetic algorithm, there are coding and fitness function, which is very important to be well adapted to the nature of a particular problem. It has been said that the usual binary encoding or over a large alphabet cardinality. The most convenient is that the relationship between the genetic codes and solutions to the problem is bijectiv mapping. Then it is possible that the application of genetic operators in a certain age to get called *incorrect specimen*, ie. individuals whose genetic code does not correspond to any solution. Overcoming this problem is possible in several ways. One possibility is to assign any such individuals as the fitness function value is zero, so that the operator applying for selection to eliminate these individuals. This approach has proven to be suitable only if the ratio of the number of incorrect and correct individuals in the population is too large, which in practice often not the case. It is possible, however, incorrect inclusion of individuals in the population by the individuals assigned to each incorrect value penalty function. The aim is unfair to individuals and get a chance to participate in the crossing, but to be discriminated against on the correct individual. Care should be taken on how to balance the value of penalty function, because too small values can lead to a genetic algorithm that some of the incorrect code for a declaration of the solution, while, on the other hand, excessive punishment can cause loss of useful information from the incorrect individuals. There is another way to solve this problem - which is to improve specimen be unfair to make them correct or incorrect that each individual is replaced correctly.

Calculating the fitness function is possible in several ways. Some of these methods are direct download, linear scaling, interval scaling, sigma truncation, etc..

Since the selection is directly related to the fitness function, the basic way to implement this genetic operator is the simple roulette selection. This method uses a distribution where the probability of selection proportional to its adaptation to the individual. Individuals involved with the chances of roulette in accordance with them, pass or not pass the process of creating a new generation. The lack of a simple roulette selection is the possibility of premature convergence due to the gradual prevalence of highly adapted individuals in the population that do not correspond to the global optimum.

To avoid this problem can be used ranking selection based on genetic codes, according to their adaptability. Fitness function is equal to the individual a range of pre-specified number of ranks, and only depend on the position of individuals in the population. It can be used linearly, as well as other forms of ranking.

Another form of selection is the tournament selection. When tournament selection is randomly generated subsets of the N individuals (N is the pre-set number), then in each subset, the principle of the tournament, selects the best individual that participates in the creation of a new generation. Usually the problem is the choice of N so as to reduce the adverse effects of stochastic, so that better and more diverse genetic material passed to the next generation. In cases where the size is perfect tournament is not an integer, has proved successful fine-graded tournament selection (FGTS). A detailed description of these and other types of selection and its theoretical aspects can be found in [6]. Application of fine-graded tournament selection and comparison with other practices in the selection of operators are given in [7], [8], [9].

The process of exchange of genetic material between individuals of the parents, in order to form new offspring individuals, is performed by the crossover operator. The most common operators are one-point crossover, two-point crossover, multi-point and a uniform crossover, and can also be used for crossover mixing, reduced surrogate crossover, crossover with the mother, intermediate crossover, as well as the linear intersection.

Intersection operator, which is implemented in a simple genetic algorithm, the one-point crossover. In one-point crossover, so determined crossing position. All genes from the predetermined position, change position so that each parental pair created two offspring. In two-point crossover two positions are set and is the exchange of genetic material between the parents and two positions.

As for the uniform crossover, it should be noted that for each parental pair determines a binary string of length the same as the genetic parents. This range is called the mask. Sharing genes is performed only on those positions where the mask is 0, while in positions where there is one, the parents retain their genes.

Mutation operator is considered one of the most important and as such it can decisively influence the operation of genetic algorithm. If a genetic algorithm using binary encoding and the population of individuals not incorrect, it is usually implemented by a simple mutation operator that runs through the individual genes and the genetic code for each check whether or not mutated. The probability of mutation is pre-set p_{mut} small size, usually taken from the interval [0.001,0.01]. Simple mutation is sometimes possible to apply over a binary number - the mask, which is randomly generated for each individual, and carries information about the position in which the genetic code results in a change of genes.

When the gene encoding algorithm used whole or real numbers (floating point), it was necessary to develop other concepts of mutation, which was done. These are the replacement of genes randomly selected number (random replacement), add or subtract a small value (creep), multiply the number close to one (geometric creep) and so on. For both creep mutation operator required values are random and can have a uniform, exponential, Gaussian or binomial distribution (see [1], [2]).

In some cases it is useful to genes, depending on the position in the genetic code, have different levels of mutation. In this regard it is particularly important concept of frozen gene. Namely, if in a position of the genetic code in all or most of the population, the same gene, it is useful that the gene mutation has a higher level than the rest of the genetic code. This concept is used to restore lost diversity of genetic material, and these genes are called frozen.

Will the application of genetic algorithms have a success depends largely on the choice of replacement policy generation. Some of the most important policy of the replacement generation: generational genetic algorithm, genetic algorithm stationary and elitist strategy. Of course, it is possible to combine these principles.

Where the generational genetic algorithm, then apply to all individuals all the genetic operators, ie. there are no privileged individuals are going into the next generation, or individuals who go directly to the selection process.

On the contrary, stationary genetic algorithm favors the best individuals in the population so as to them shall not apply operator selection, but they go directly to the next stage, while the other applies the selection of individuals and they come to the remaining places.

Elitist strategy provides a direct passage into the next generation of one of the best individuals. These individuals do not apply to operators of selection, crossover and mutation. By applying genetic operators to the individuals of the population remaining seats are filled by the next generation.

This approach leaves room for another possible improvement of the genetic algorithm, which is caching. As an elite individuals pass from generation to generation unchanged, it and its value remains unchanged. Therefore, it would be useful to individuals elitinih value is remembered, rather than constantly calculated, saving the time required for their computation. This process is called caching, and more detail is in [12] and [11].

Namely, the calculated value of the objective function of individuals are stored in so-called. Hess-row table, which uses CRC codes that are assigned to individuals in population. If, during operation of the genetic algorithm, obtained through the same genetic code, then the objective function value is taken from a hash-table, through the CRC code.

Input for the algorithm for slving the MDBCS problem is undirected graph $G = (V, E)$, weight function (weighting function) $w: E \mapsto \mathbb{R}^+$and integer $d \geq 2$. Each vertices of the graph encode the zero (0), if the condition (2) does not satisfy, and one (1), otherwise (we use that is, a binary encoding of individuals). If we find the node that has more than d vertices, then vertices of the node does not count, that is. set value 0.

In order to get connected components (or component connection), we apply a search in width. If the graph is only one component connection, then this is the end, that is, the graph is connected and calculate the sum of all vertices of the weight of the component. But if more than one connected components, then we take them in order, first and second, second and third, and so on, and look for the largest vertice of the weight (or cycle if it exists) that connects the two components. Then take all the vertices of the shortest path from the original graph, and the count only those with a degree $\leq d$. The process continues until they connect all components of relationship. Finally, we add all of this and get $\sum_{e \in E'} w(e)$.

Genetic operators, which are used here, are fine-graded tournament selection, a one-point crossover, a simple mutations with the frozen gene and caching techniques. Genetic algorithm is coded in the programming language C.

For checking the results of the implemented genetic algorithm based on mathematical models (1) - (7), we used the software package CPLEX. Genetic algorithm is tested on the test-examples reached the same values as the CPLEX program, and that the execution time was short in both cases, data on execution times not are presented.

As there are no standard instances of MDBCS problem, and the instance for KCT problem (k-cardinality tree problem), containing the nodes and weights that are appropriate for the considered problem, the necessary adjustments in accordance with the input of a genetic algorithm, are used in testing of the genetic algorithm for graphs with a large number of vertices and nodes.

Stopping criterion of the algorithm is the maximum number of generations 5000 or up to 2000 generations without improving the objective function value.

III. TEST EXAMPLES

Here are a few tables where the columns of labels, respectively, represent:

- instance name that contains the dimension of the input graph, an *Instance name*;
- cardinality of the set of nodes, n;
- cardinality of the set of vertices, m;
- integer $d \geq 2, d$;
- best solution obtained by genetic algorithm, GA_{best};
- average time t (in seconds) to calculate the best value;
- t_{tot}total time (in seconds) to complete a genetic algorithm;
- total number of generation, *gen*;
- average value of using caching, *cache*.

The Tables 1 - 5 shows the results obtained by testing the genetic algorithm for instances with a large number of vertices and nodes, for different values of d.

IV. CONCLUSION

The problem of finding the maximum degree of limitation associated subgraph is very interesting, and then proposed a genetic algorithm can be the basis for further research and improvement. In addition, this class of combinatorial problems has been studied extensively and due to the application in the design of networks of networks and routing algorithms.

Some of the directions of further enlargement and improvement of the results could be:

1) the development of exact methods based on integer programming;

2) modification of metaheuristic described - genetic algorithm for solving similar problems on graphs;

3) obtaining new results from graph theory developed using the implementation.

REFERENCES

[1] Beasley D., Bull D. R., Martin R. R., „ An Overview of Genetic Algorithms, Part1," Research Topics. University Computing; 1993, Vol. 15, No. 2, p. 58-69.

[2] Beasley D., Bull D. R., Martin R. R., „ An Overview of Genetic Algorithms, Part2," Research Topics. University Computing; 1993, Vol. 15, No. 4, p. 170-181.

[3] Bogdanović M., Rešavanje problema maksimalnog ograničenja stepena podgrafova u računarstvu, kao prilog teoriji grafova, Doktorska disertacija. Univerzitet u Beogradu, Matematički fakultet; 2010, PhD Thesis (in Serbian).

[4] Bogdanović M., „ An ILP formulation for the maximum degree-bounded connected subgraph problem", Computers & Mathematics with Applications; 2010, 59, No.9, p. 3029-3038.

[5] Bogdanović M. „On some basic concepts of genetic algorithms as a meta-heuristic method for solving of optimization problems", A Journal of Software Engineering and Applications; 2011, Vol. 4, No. 8, pp. 482-486, doi: 10.4236/jsea.2011.48055. Website: http://www.scirp.org/journal/jsea

[6] Filipović V., Predlog poboljšanja operatora turnirske selekcije kod genetskih algoritama, Magistarski rad. Univerzitet u Beogradu, Matematički fakultet; 1998. MsThesis (in Serbian)

[7] Filipović V., Kratica J., Tošić D., Ljubić I., „Fine Grained Tournament Selection for the Simple Plant Location Problem", Proceedings on the 5th Online World Conference on Soft Computing Methods in Industrial Application – WSC5; 2000, p. 152-158.

[8] Filipović V., Tošić D., Kratica J., "Experimental Results in Applying of Fine Grained Tournament Selection", Proceedings of the 10th Congress of Yugoslav Mathematicians Belgrade, 21.-24.01.; 2001, p. 331-336.

[9] Filipović, V., "Fine-Grained Tournament Selection Operator in Genetic Algorithms", Computing and Informatics; 2003. 22(2), p.143-161.

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

[11] Kratica J., „ Improvement of Simple Genetic Algorithm for Solving the Uncapacitated Warehouse Location Problem", Advances in Soft Computing – Engineering Design and Manufactining, R. Roy, T. Furuhachi and P. K. Chawdhry (Eds), Springer-Verlang London Limited; 1999, p. 390-402.

[12] Kratica J., Paralelizacija genetskih algoritama za rešavanje nekih NP-kompletnih problema, Doktorska disertacija. Matematički fakultet, Beograd; 2000, PhD Thesis (in Serbian).

TABLE 1.

Instance name	*n*	*m*	*d*	GA_{best}	$t[s]$	$t_{tot}[s]$	*gen*	*cache [%]*
Proba10-13	**10**	**13**	**2**	**60**	**0.0001**	**0.50**	**2003**	**97.94716**
Proba10-13	**10**	**13**	**3**	**71**	**0.0001**	**0.53**	**2004**	**98.11958**
Proba10-13	**10**	**13**	**4**	**74**	**0.0001**	**0.48**	**2005**	**98.19920**
Proba10-13	***10***	***13***	***5***	***75***	***0.0001***	***0.50***	***2004***	***98.18934***
Proba10-13	***10***	***13***	***6***	***75***	***0.0001***	***0.48***	***2004***	***98.19934***

TABLE 2.

Instance name	*n*	*m*	*d*	GA_{best}	$t[s]$	$t_{tot}[s]$	*gen*	*cache [%]*
w40-40	**40**	**40**	**2**	**576**	**0.02**	**0.53**	**2042**	**75.21076**
w40-40	**40**	**40**	**3**	**768**	**0.09**	**0.64**	**2298**	**80.43894**
w40-40	**40**	**40**	**4**	**906**	**0.01**	**0.55**	**2040**	**82.20656**
w40-40	**40**	**40**	**5**	**1013**	**0.05**	**0.59**	**2114**	**82.38545**
w40-40	**40**	**40**	**6**	**1069**	**0.01**	**0.55**	**2048**	**82.72843**
w40-40	**40**	**40**	**7**	**1111**	**0.01**	**053**	**2037**	**85.22745**
w40-40	***40***	***40***	***8***	***1120***	***0.0001***	***0.52***	***2026***	***85.13061***
w40-40	***40***	***40***	***9***	***1120***	***0.0001***	***0.52***	***2026***	***85.13061***
h44-44	**44**	**44**	**2**	**85**	**0.08**	**0.67**	**2032**	**57.63736**
h44-44	**44**	**44**	**3**	**203**	**0.02**	**0.53**	**2050**	**79.11934**
h44-44	**44**	**44**	**4**	**292**	**0.05**	**0.56**	**2137**	**77.93271**
h44-44	**44**	**44**	**5**	**343**	**0.03**	**0.50**	**2112**	**81.25106**
h44-44	**44**	**44**	**6**	**371**	**0.01**	**0.55**	**2042**	**83.19511**
h44-44	***44***	***44***	***7***	***377***	***0.01***	***0.53***	***2038***	***85.13866***
h44-44	***44***	***44***	***8***	***377***	***0.01***	***0.53***	***2038***	***85.13866***
w65-65	**65**	**65**	**2**	**283**	**0.25**	**1.11**	**2456**	**43.06466**
w65-65	**65**	**65**	**3**	**787**	**0.42**	**1.22**	**3004**	**52.15896**
w65-65	**65**	**65**	**4**	**1200**	**0.47**	**1.28**	**3101**	**55.82088**
w65-65	**65**	**65**	**5**	**1488**	**0.05**	**0.75**	**2093**	**68.62977**
w65-65	**65**	**65**	**6**	**1644**	**0.34**	**1.03**	**2913**	**71.04870**
w65-65	**65**	**65**	**7**	**1744**	**0.03**	**0.72**	**2060**	**72.59913**
w65-65	**65**	**65**	**8**	**1792**	**0.03**	**0.70**	**2058**	**76.46288**
w65-65	***65***	***65***	***9***	***1801***	***0.05***	***0.72***	***2077***	***75.60385***
w65-65	***65***	***65***	***10***	***1801***	***0.05***	***0.72***	***2077***	***75.60385***

TABLE 3.

Instance name	*n*	*m*	*d*	GA_{best}	$t[s]$	$t_{tot}[s]$	*gen*	*cache [%]*
h118-118	**118**	**118**	**2**	**127**	**0.01**	**1.33**	**2007**	**33.67463**
h118-118	**118**	**118**	**3**	**601**	**0.69**	**2.00**	**2999**	**39.21786**
h118-118	**118**	**118**	**5**	**1235**	**2.59**	**3.34**	**5000**	**44.09634**
h118-118	**118**	**118**	**6**	**1663**	**3.06**	**3.19**	**5000**	**49.11733**
h118-118	**118**	**118**	**7**	**1958**	**0.41**	**1.48**	**2648**	**59.18974**
h118-118	**118**	**118**	**8**	**2058**	**0.11**	**1.17**	**2143**	**61.19571**
h118-118	***118***	***118***	***9***	***2091***	***0.09***	***1.17***	***2117***	***61.18774***
h118-118	***118***	***118***	***10***	***2091***	***0.09***	***1.17***	***2117***	***61.18774***
w130-130	**130**	**130**	**2**	**295**	**0.02**	**1.38**	**2017**	**34.75842**
w130-130	**130**	**130**	**3**	**1030**	**1.30**	**2.70**	**3788**	**34.68636**
w130-130	**130**	**130**	**4**	**1657**	**0.31**	**1.75**	**2406**	**36.35866**
w130-130	**130**	**130**	**5**	**2084**	**0.88**	**2.28**	**3195**	**39.45466**
w130-130	**130**	**130**	**6**	**2323**	**0.64**	**2.08**	**2877**	**38.50201**
w130-130	**130**	**130**	**7**	**2481**	**0.13**	**1.50**	**2152**	**42.08631**
w130-130	**130**	**130**	**8**	**2582**	**0.36**	**1.67**	**2488**	**46.23685**
w130-130	***130***	***130***	***9***	***2591***	***0.11***	***1.41***	***2117***	***46.92453***
w130-130	***130***	***130***	***10***	***2591***	***0.11***	***1.41***	***2117***	***46.92453***
h183-183	**183**	**183**	**2**	**169**	**0.01**	**1.83**	**2019**	**32.26805**
h183-183	**183**	**183**	**3**	**1000**	**2.47**	**4.36**	**4525**	**32.48498**
h183-183	**183**	**183**	**4**	**2056**	**2.20**	**4.16**	**4167**	**35.62782**
h183-183	**183**	**183**	**5**	**3007**	**2.77**	**4.66**	**4780**	**40.11583**
h183-183	**183**	**183**	**6**	**3776**	**0.83**	**2.48**	**2892**	**50.47738**
h183-183	**183**	**183**	**7**	**4147**	**2.09**	**3.75**	**4392**	**51.32014**
h183-183	**183**	**183**	**8**	**4343**	**1.59**	**3.23**	**3829**	**51.77192**
h183-183	**183**	**183**	**9**	**4425**	**0.27**	**1.92**	**2232**	**51.54989**
h183-183	***183***	***183***	***10***	***4457***	***0.38***	***2.00***	***2367***	***51.83713***
h183-183	***183***	***183***	***11***	***4457***	***0.39***	***2.02***	***2367***	***51.83713***

TABLE 4.

Instance name	*n*	*m*	*d*	GA_{best}	$t[s]$	$t_{tot}[s]$	*gen*	*cache [%]*
h212-212	**212**	**212**	**2**	**156**	**0.22**	**2.30**	**2179**	**32.14757**
h212-212	**212**	**212**	**3**	**1203**	**1.52**	**3.75**	**3230**	**33.58119**
h212-212	**212**	**212**	**4**	**2624**	**4.09**	**5.84**	**5000**	**33.34679**
h212-212	**212**	**212**	**5**	**3959**	**4.95**	**5.69**	**5000**	**37.96402**
h212-212	**212**	**212**	**6**	**4902**	**2.72**	**4.69**	**4645**	**47.71343**
h212-212	**212**	**212**	**7**	**5376**	**2.55**	**4.50**	**4502**	**49.54007**
h212-212	**212**	**212**	**8**	**5571**	**1.33**	**3.25**	**3279**	**49.15052**
h212-212	**212**	**212**	**9**	**5654**	**2.20**	**4.09**	**4225**	**50.38174**
h212-212	***212***	***212***	***10***	***5686***	***0.38***	***2.27***	***2291***	***48.96513***
h212-212	***212***	***212***	***11***	***5686***	***0.36***	***2.25***	***2291***	***48.96513***
h44-48	**44**	**48**	**2**	**177**	**0.27**	**0.86**	**2826**	**61.00530**
h44-48	**44**	**48**	**3**	**335**	**0.48**	**1.22**	**3799**	**78.49343**
h44-48	**44**	**48**	**4**	**405**	**0.05**	**0.61**	**2110**	**72.81590**
h44-48	**44**	**48**	**5**	**452**	**0.19**	**0.77**	**2679**	**73.40791**
h44-48	**44**	**48**	**6**	**477**	**0.05**	**0.59**	**2147**	**77.68186**
h44-48	**44**	**48**	**7**	**484**	**0.02**	**0.55**	**2045**	**78.76856**
h44-48	***44***	***48***	***8***	***489***	***0.01***	***0.55***	***2032***	***82.67125***
h44-48	***44***	***48***	***9***	***489***	***0.01***	***0.55***	***2032***	***82.67125***
w70-76	**70**	**76**	**2**	**625**	**0.11**	**0.97**	**2225**	**44.62029**
w70-76	**70**	**76**	**3**	**1307**	**0.31**	**1.20**	**2664**	**50.06899**
w70-76	**70**	**76**	**4**	**1732**	**0.09**	**0.89**	**2157**	**62.85833**
w70-76	**70**	**76**	**5**	**1991**	**0.06**	**0.81**	**2096**	**69.20438**
w70-76	**70**	**76**	**6**	**2109**	**0.08**	**0.80**	**2099**	**69.92674**
w70-76	**70**	**76**	**7**	**2159**	**0.66**	**1.42**	**3702**	**69.68583**
w70-76	**70**	**76**	**8**	**2203**	**0.03**	**0.80**	**2068**	**70.37373**
w70-76	***70***	***76***	***9***	***2224***	***0.03***	***0.75***	***2064***	***72.95791***
w70-76	***70***	***76***	***10***	***2224***	***0.03***	***0.77***	***2064***	***72.95791***

TABLE 5.

Instance name	*n*	*m*	*d*	GA_{best}	$t[s]$	$t_{tot}[s]$	*gen*	*cache [%]*
proba12-17	**12**	**17**	**2**	**118**	**0.02**	**0.53**	**2010**	**95.20616**
proba12-17	***12***	***17***	***3***	***128***	***0.0001***	***0.55***	***2011***	***96.62761***
proba12-17	***12***	***17***	***4***	***128***	***0.0001***	***0.52***	***2011***	***96.62761***
g15-4-01	**15**	**20**	**2**	**326**	**0.0001**	**0.52**	**2011**	**94.55214**
g15-4-01	***15***	***20***	***3***	***335***	***0.0001***	***0.53***	***2011***	***95.36941***
g15-4-01	***15***	***20***	***4***	***335***	***0.0001***	***0.52***	***2011***	***95.36941***
liter34-39	**34**	**39**	**2**	**282**	**0.0001**	**0.56**	**2018**	**67.02524**
liter34-39	**34**	**39**	**3**	**349**	**0.01**	**0.53**	**2028**	**84.96898**
liter34-39	**34**	**39**	**4**	**369**	**0.09**	**0.66**	**2363**	**81.58072**
liter34-39	***34***	***39***	***5***	***371***	***0.01***	***0.53***	***2033***	***84.18566***
liter34-39	***34***	***39***	***6***	***371***	***0.01***	***0.53***	***2033***	***84.18566***
h69-69	**69**	**69**	**2**	**382**	**0.13**	**0.95**	**2248**	**49.79209**
h69-69	**69**	**69**	**3**	**572**	**0.28**	**1.03**	**2680**	**62.79687**
h69-69	**69**	**69**	**4**	**690**	**0.33**	**1.03**	**2781**	**69.99425**
h69-69	**69**	**69**	**5**	**752**	**0.16**	**0.84**	**2351**	**71.30161**
h69-69	***69***	***69***	***6***	***771***	***0.03***	***0.72***	***2064***	***74.50798***
h69-69	***69***	***69***	***7***	***771***	***0.03***	***0.72***	***2064***	***74.50798***
h148-152	**148**	**152**	**2**	**2107**	**0.77**	**2.45**	**2829**	**38.28884**
h148-152	**148**	**152**	**3**	**3470**	**2.16**	**3.66**	**4685**	**48.01664**
h148-152	**148**	**152**	**4**	**3811**	**0.56**	**1.97**	**2688**	**53.01078**
h148-152	**148**	**152**	**5**	**3931**	**0.17**	**1.61**	**2178**	**51.45988**
h148-152	***148***	***152***	***6***	***3941***	***0.19***	***1.56***	***2191***	***54.73291***
h148-152	***148***	***152***	***7***	***3941***	***0.19***	***1.56***	***2191***	***54.73291***