Taylor & Francis
Taylor & Francis Group

Automatika

Journal for Control, Measurement, Electronics, Computing and Communications

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/taut20

An evolutionary algorithm for the robust maximum weighted independent set problem

Ana Klobučar & Robert Manger

To cite this article: Ana Klobučar & Robert Manger (2020) An evolutionary algorithm for the robust maximum weighted independent set problem, Automatika, 61:4, 523-536, DOI: 10.1080/00051144.2020.1789364

To link to this article: https://doi.org/10.1080/00051144.2020.1789364

Published online: 21 Jul 2020.

AUTOMATIKA
2020, VOL. 61, NO. 4, 523–536
https://doi.org/10.1080/00051144.2020.1789364

REGULAR PAPER

OPEN ACCESS Check for updates

An evolutionary algorithm for the robust maximum weighted independent set problem

Ana Klobučar [a] and Robert Manger [b]

[a]Faculty of Mechanical Engineering and Naval Architecture, University of Zagreb, Zagreb, Croatia; [b]Department of Mathematics, Faculty of Science, University of Zagreb, Zagreb, Croatia

ABSTRACT

This work deals with the robust maximum weighted independent set problem, i.e. finding a subset of graph vertices that are not adjacent to each other and whose sum of weights is as large as possible. Uncertainty in problem formulation is restricted to vertex weights and expressed explicitly by a finite set of scenarios. Three criteria of robustness are considered: absolute robustness (max-min), robust deviation (min-max regret), and relative robustness (relative min-max regret). Since the conventional maximum weighted independent set problem is already NP-hard, finding the exact solution of its robust counterpart should obviously have a prohibitive computational complexity. Therefore, we propose an approximate algorithm for solving the considered robust problem, which is based on evolutionary computing and on various crossover and mutation operators. The algorithm is experimentally evaluated on appropriate problem instances. It is shown that satisfactory solutions can be obtained for any of the three robustness criteria in reasonable time.

ARTICLE HISTORY
Received 27 September 2019
Accepted 26 June 2020

KEYWORDS
Robust optimization; maximum weighted independent set; approximation; evolutionary algorithm; complexity

1. Introduction

A conventional optimization problem consists of maximizing or minimizing an objective function over a set of feasible solutions that satisfy given constraints. However, in real-life situations, input parameters that specify a particular problem instance are often uncertain or subject to change, since they may be influenced by some unpredictable future circumstances. We can try to somehow estimate or approximate those parameters, but such estimations could easily lead to an inferior or even infeasible solution. Instead of ignoring uncertainty, it is much better to handle it by using an appropriate mathematical model.

A state-of-the art approach to deal with the mentioned uncertainty is called robust optimization [1,2]. According to that approach, a finite or infinite set of scenarios is defined. The scenarios should capture uncertainty in problem parameters in some way. Only those solutions are considered that are feasible for all scenarios. As the optimal solution in the robust sense, the one is chosen whose worst behaviour over the whole set of scenarios happens to be the best among all solutions. Such solution does not need to be really optimal in the conventional sense, but it is chosen in order to be acceptable even in the most adverse circumstances.

In this paper, we consider the maximum weighted independent set problem, which is posed in a graph whose vertices are given weights. In its conventional form, the problem consists of finding a subset of graph vertices that are not adjacent to each other and whose sum of weights is as large as possible. Our focus is on several robust variants of the same problem, where graph structure is fixed but vertex weights are uncertain. Such uncertainty is expressed by a finite set of scenarios.

It is well known [3] that the conventional maximum weighted independent set problem is already NP-hard. Thus finding exact solutions of its robust variants should be even harder. Consequently, we do not expect that real-life problem instances could be solved to optimality in reasonable time. Instead, we believe that the only practical way of solving such instances is by using approximate algorithms.

The aim of this paper is to demonstrate that robust variants of the maximum weighted independent set problem can be solved by approximate algorithms that are accurate and fast enough for real-life applications. To fulfil this aim, the paper proposes an algorithm for solving the considered robust problem variants, which is based on evolutionary computing and on various original evolutionary operators (crossovers, mutations, recovery). The proposed algorithm is experimentally evaluated on appropriate problem instances. Through such evaluation, the best-performing combinations of crossovers and mutations are identified.

CONTACT Robert Manger manger@math.hr Department of Mathematics, Faculty of Science, University of Zagreb, Bijenička cesta 30, Zagreb, Croatia

Building our algorithm on a heuristic such as evolutionary computing (and not on a more "exact" method) is motivated by the following facts. It is true that the considered robust problem can be formulated as an integer linear programming problem and then processed by a general-purpose optimization software. But due to NP-hardness, such approach does not work for larger problem instances. Moreover, according to our preliminary investigations, approximate solving based on a relaxed linear-programming model would not work either, since it would produce non-integer solutions that cannot easily be rounded to integer solutions. Thus among various approximate algorithms, evolutionary computing and similar heuristics seem to be the best choice.

Apart from this introduction, the rest of the paper is organized as follows. Section 2 lists all necessary definitions and preliminaries. Section 3 gives examples of real-world applications of the considered problem and its robust variants. Section 4 presents the overall structure and properties of our evolutionary algorithm for solving the robust variants. Sections 5–7 describe in more detail the concrete operators used by the algorithm, i.e. four crossovers, one recovery operator and five mutations, respectively. Section 8 reports on experimental evaluation, i.e. on the performed tests and their results. The final Section 9 gives conclusions.

2. Definitions and preliminaries

The foundations of robust optimization have been laid out in the seminal works [2,4–6]. More recent surveys and some general results can be found in [1,7–9]. The book [2] gives a suitable framework for robust *discrete* optimization.

According to [2], uncertainty in problem parameters should be captured by a finite and explicitly given set of *scenarios*. Each scenario specifies a possible combination of parameter values. As mentioned before, only those solutions are considered that are feasible for all scenarios. The behaviour of any considered solution under any scenario is measured according to some *criterion of robustness*. Then, the so-called *robustly optimal* solution in chosen as the one whose worst behaviour, measured over all scenarios, is the best possible.

The framework from [2] obviously allows many options. For instance, the set of uncertain parameters can be more or less extensive. Also, the behaviour of a solution under a scenario can be measured by different criteria of robustness. Thus for the same conventional (non-robust) optimization problem one can construct several robust problem variants, which can be more or less difficult to solve. There are three popular criteria of robustness, and according to [2], they are called absolute robustness, robust deviation, and relative robust deviation. In some other publications, e.g.[1], the same criteria are referred to as max-min (min-max), min-max regret, and relative min-max regret, respectively.

Let now S denote the set of all scenarios, and suppose that each scenario is encoded as an n-tuple, where n is the number of input parameters. Let X be a feasible solution, $F(X, s)$ the (conventional) objective-function value for solution X under scenario s, $F^*(s)$ the optimal (conventional) solution value for scenario s, and Φ the set of all solutions that are feasible for all scenarios. Suppose also that the considered conventional problem is a maximization problem, and that for any scenario s the value $F^*(s)$ is not zero. Then the three previously mentioned robustness criteria are defined in the following way.

- *Absolute robust solution (max-min).* X_A is a feasible solution whose minimum objective function value, measured over all scenarios, is as large as possible, i.e.

$$\mathrm{opt}_A = \max_{X\in\Phi} \min_{s\in S} F(X,s) = \max_{X\in\Phi} F_A(X) = F_A(X_A).$$

  The robust objective function used here (being maximized) is

$$F_A(X) = \min_{s\in S} F(X,s). \tag{1}$$

- *Robust deviation solution (min-max regret).* X_D is a feasible solution whose maximum deviation from the conventional optimum, measured over all scenarios, is as small as possible, i.e.

$$\begin{aligned}\mathrm{opt}_D &= \min_{X\in\Phi} \max_{s\in S} \Big(F^*(s) - F(X,s)\Big)\\ &= \min_{X\in\Phi} F_D(X) = F_D(X_D).\end{aligned}$$

  The corresponding robust objective function (now being minimized) is

$$F_D(X) = \max_{s\in S} \Big(F^*(s) - F(X,s)\Big). \tag{2}$$

- *Relative robust deviation solution (relative min-max regret).* X_R is a feasible solution whose maximum relative deviation from the conventional optimum, measured over all scenarios, is as small as possible, i.e.

$$\begin{aligned}\mathrm{opt}_R &= \min_{X\in\Phi} \max_{s\in S} \left(\frac{F^*(s) - F(X,s)}{F^*(s)}\right)\\ &= \min_{X\in\Phi} F_R(X) = F_R(X_R).\end{aligned}$$

  The corresponding robust objective function (again being minimized) is now

$$F_R(X) = \max_{s\in S} \left(\frac{F^*(s) - F(X,s)}{F^*(s)}\right). \tag{3}$$

As already announced, in this paper, we study the maximum weighted independent set problem. Its conventional variant is very well known and treated in many textbooks, e.g.[10,11]. We give here some basic definitions for convenience.

- Let $G = (V, E)$ be an undirected graph, where V is the set of vertices and E the set of edges. An *independent set* of G is a subset X of V such that no two vertices in X are adjacent (connected by an edge from E).
- Let $G = (V, E)$ be an undirected graph whose vertices have weights. Suppose that all weights are positive real numbers. A *maximum weighted independent set* of G is an independent set of G whose sum of vertex weights is as large as possible.
- The problem of finding a maximum weighted independent set in a given weighted graph is called the (conventional) *maximum weighted independent set problem* (the *MWIS problem*).

In this paper, we consider robust variants of the above defined MWIS problem. Those variants are constructed according to the framework from [2]. Thereby the graph structure is assumed to be stable, only vertex weights are considered as uncertain. Consequently, each scenario is simply an n-tuple of weights, where n is the number of vertices in the graph.

So our robust problem can be defined as follows. For a given graph $G = (V, E)$ and a finite set S of scenarios for its vertex weights, find an independent set X_A (or X_D or X_R) that satisfies one of the previously listed robustness criteria. Or in other words, find an independent set that optimizes the robust objective function (1) (or (2) or (3)). Such problem is called the *robust maximum weighted independent set problem* (the *RMWIS problem*). Depending on the chosen criterion, there are three variants of the RMWIS problem.

It is important to note that the conventional MWIS problem can be written formally as an integer linear programming (ILP) problem, as shown below. Such definition is necessary in order to be able to solve smaller problem instances by general-purpose optimization packages, e.g. by IBM ILOG CPLEX Optimization Studio [12].

$$\begin{aligned} &\text{maximize} \sum_{i=1}^{n} w_i x_i \\ &\text{subject to:} \\ &x_i + x_j \leq 1, \quad \text{for all } (v_i, v_j) \in E \\ &x_i \in \{0, 1\}, \quad \text{for all } i = 1, \ldots, n. \end{aligned}$$

Here, n is again the number of vertices in our graph G, i.e. $n = |V|$. The symbols v_i, $i = 1, \ldots, n$, denote vertices, and w_i, $i = 1, \ldots, n$, are their weights. The decision variable x_i corresponds to v_i and it is equal 1 if and only if v_i is included in the solution. The first condition assures that no two vertices within the solution are adjacent. We maximize the sum of weights among all feasible solutions. So the objective function can also be written as:

$$F(X) = \sum_{v_i \in X} w_i,$$

where X is the set of chosen vertices, i.e. those vertices v_i whose corresponding values x_i are 1. In the robust case, the objective function at scenario s is the following:

$$F(X, s) = \sum_{v_i \in X} w_i^s. \tag{4}$$

Here w_i^s denotes the weight of vertex v_i under scenario s.

Similarly as the conventional MWIS problem, our three variants of the robust MWIS problem can also be formulated in terms of ILP and hopefully solved by general-purpose optimization packages. Indeed, for the absolute robust MWIS problem (max-min) the ILP formulation looks as follows:

$$\begin{aligned} &\text{maximize } y \\ &\text{subject to:} \\ &\sum_{i=1}^{n} w_i^s x_i \geq y, \quad \text{for all } s \in S \\ &x_i + x_j \leq 1, \quad \text{for all } (v_i, v_j) \in E \\ &x_i \in \{0, 1\}, \quad \text{for all } i = 1, \ldots, n. \end{aligned} \tag{5}$$

For the robust deviation MWIS problem (min-max regret), the ILP formulation is the same as above, except that y is minimized and the first constraint is replaced with:

$$F^*(s) - \sum_{i=1}^{n} w_i^s x_i \leq y, \quad \text{for all } s \in S \tag{6}$$

The ILP formulation for the relative robust deviation MWIS problem (relative min-max regret) is again the same, except y being minimized and the first constraint being:

$$\sum_{i=1}^{n} w_i^s x_i \geq (1 - y) F^*(s), \quad \text{for all } s \in S \tag{7}$$

At this moment, there are only few publications on the RMWIS problem found in the literature, i.e.[13–16]. The cited papers are concerned with complexity issues, restrictions to special types of graphs, or relations to other problems. We are not aware of any paper dealing with reasonably efficient algorithms for the RMWIS problem on general graphs. So our paper tries to fill this gap. On the other hand, there is a fair number of publications where the conventional MWIS problem is solved by exact or approximate algorithms. For

instance, [17] presents an exact algorithm for the MWIS problem, [18,19] develop greedy heuristics, [20,21] propose local-search heuristics, while [22,23] study genetic algorithms.

At the end of this section, let us say something more about computational complexity of the considered problems. The proof that the conventional MWIS problem is NP-hard can be found in many books including also the classical monograph [3]. Since the MWIS problem can be regarded as a special case of the RMWIS problem (having only one scenario), it is clear that the RMWIS problem must also be NP-hard.

3. Applications

Although the considered MWIS problem looks rather abstract and artificial, it still can be applied to many practical situations. This is true not only for its conventional variant but also for its robust variants. The most notorious application is *facility location* [24], where locations for certain facilities (e.g. warehouses) should be chosen, so that some kind of profit is maximized and the facilities are not too close one to another. Also very interesting application is described in [25]. It deals with *selection of programme slots of television channels* for giving an advertisement. One would like to choose the slots that do not overlap in time, while attracting as many viewers as possible. In this section, we will describe in more detail an application dealing with *labeling of a digital map*. It is a slightly modified version of the application from [26].

We consider a digital map of a geographical region, which is used within a mobile app. The map stores positions of many locations, so-called *points of interests* (POI-s). Each POI is assigned a positive number interpreted as its importance. The POI-s are divided into categories, e.g. restaurants, cinemas, shopping malls, etc. The user observes the map through a rectangular window. It means that, at any moment, only a part of the map is visible. The user can move the window in all directions, or use zoom-in or zoom-out operations to shrink or extend the visible area of the map.

The main purpose of our app is to inform the user about existence of certain POI-s. Therefore, the app tries to show the positions of all POI-s that reside within the currently displayed area. The user can select only particular categories of POI-s to be shown, e.g. only restaurants, or only cinemas, or only restaurants and cinemas, etc.

Our app marks the position of a POI by displaying its label (i.e. a text or icon) at the appropriate place within the display. Unfortunately, it can happen that two or more labels overlap if the corresponding POI-s are too close or if the displayed area of the map is too big due to zooming-out. Overlapping of labels should be avoided since it spoils readability of the display. In order to prevent overlapping, our app should show only the labels of more important POI-s and hide the remaining labels.

So we are confronted with the following problem: which of the overlapping labels should be shown, and which should be hidden? The problem can be solved by constructing the so-called *conflict graph*. Vertices of that graph correspond to POI-s that belong to the currently displayed area and the currently selected categories. The weight of a vertex is equal to the importance measure of the corresponding POI. Any two vertices are connected by an edge if and only if the corresponding labels would overlap on the current display.

Obviously, an independent set of the conflict graph determines a set of POI-s whose labels can be displayed without overlapping. Moreover, a maximum-weight independent set specifies the set of POI-s whose total importance is as large as possible and whose labels do not overlap. Thus to decide which labels should be shown and which should be hidden, our app must solve the MWIS problem on the current conflict graph. Note that the conflict graph changes whenever the user selects different categories of POI-s, or moves the window, or zooms the displayed area. Thus the MWIS problem has to be solved many times and as quickly as possible.

Let us now discuss about uncertainty of vertex weights within the considered conflict graph. Indeed, those weights are uncertain since they reflect relative importance of particular POI-s obtained by subjective judgement. Obviously, any consistent set of importance measures can be interpreted as a scenario for weights. Ideally, each user of our app should have its own scenario. However, a more practical option is to divide users into profiles, such as young men, young women, middle-aged men, etc. Then scenarios could be associated with profiles rather than with individual users.

If the profile of the current user is known, then the associated MWIS problem instances can be solved by using the appropriate scenario, which means that the conventional (single scenario) variant of the MWIS problem is solved. On the other hand, if the user profile is not known, then all available scenarios should be taken into account. The best way how to handle more scenarios is to solve a robust variant of the MWIS problem. The obtained robust solution would be acceptable for any scenario, although not optimal. If some partial information about the user is known, e.g. that he/she is a young person, then a reduced set of scenarios could be used; the associated MWIS problem would be simpler but still robust.

4. Properties of our evolutionary algorithm

Having in mind the mentioned NP-hardness of the conventional MWIS problem, we propose an approximate algorithm for solving the RMWIS problem based on evolutionary computing. As described in many books,

e.g.[27–29], an evolutionary algorithm (EA) is a randomized computing procedure which maintains a population of *chromosomes*. Each chromosome represents a feasible solution to a given instance of an optimization problem. The population is iteratively changed, thus giving a series of population versions called generations. Change is accomplished through application of evolutionary operators, such as *crossovers* or *mutations*, which create or modify chromosomes. During the whole process, more "fit" population members have more chance to "survive", and less fit are discarded. The best chromosome in the last generation should represent a nearly optimal solution to the considered problem instance.

In order to construct a good EA for the RMWIS problem, we will consider four original crossover operators and five mutations. Also, we will propose a flexible *recovery* operator, which can modify an infeasible chromosome to make it feasible. Altogether twenty EA variants will be considered, where any of them combines one particular crossover with one particular mutation. We will apply each EA variant to each of the three RMWIS problem variants.

The mentioned operators (crossovers, mutations, recovery) will be described in more detail in subsequent sections. The remaining building blocks of our EA are realized in rather standard ways. Indeed:

- The data structure used to represent a chromosome is a list of graph vertices, which can be sorted according to various criteria. The listed vertices should constitute a feasible independent set, i.e. they should satisfy the constraint that no two of them are connected by an edge.
- Fitness of a chromosome is assessed by computing its objective-function value according to the chosen robustness criterion. Depending on the criterion, the robust objective function (1), (2) and (3), respectively, is used, combined with the conventional objective function (4).
- Selection of a "good" chromosome for crossover (or a "bad" chromosome for disposal) is based on tournament selection [27,29]. Thus a certain number of chromosomes is picked up randomly, and the most (or least) fit of them is chosen.
- New chromosomes produced by crossovers are inserted into the population as described in [30]. The procedure relies on the concept of similarity. We say that two chromosomes are similar if their robust objective-function values differ less than 1% of the best value within the population. Insertion is accomplished according to the following two rules. If there exists another chromosome in the population that is similar to the new one, then the better of those two "twins" is retained and the other one is discarded. If there is no similar chromosome, then the new one is retained and some other "bad" chromosome is selected by tournament and discarded.

Note that according to our rules, the population size always remains the same. Indeed, whenever a chromosome is inserted, another one is discarded. Note also that the whole algorithm supports so-called elitism [27,29], i.e. the best chromosome within the population is discarded only if it is replaced by an even better chromosome.

Since an EA usually requires many iterations, it is important that a particular iteration can be computed quickly. Therefore, in our crossover and recovery operators, we will mostly rely on greedy strategies. It is true that greedy decisions often lead to an unsatisfactory local optimum. However, such premature termination of computing can be avoided by efficient mutation operators.

Our greedy strategies are mostly inspired by the vertex-selecting rule from [19], which maximizes:

$$c(v_i) = \frac{w_i}{d(v_i) + 1}. \tag{8}$$

Here, w_i is again the weight of vertex v_i, and $d(v_i)$ is its degree (number of adjacent vertices). Justification for the above expression is illustrated by Figure 1. Indeed, for the shown graph and shown weights of vertices, a vertex-selecting rule which takes into consideration only weights would give an independent set of total weight 7. On the other hand, the vertex-selecting rule based on (8) would give an independent set of total weight 10. Sometimes vertices with larger weights can have many neighbours, so that better solutions can be obtained by using a larger number of more independent vertices with smaller weights.

The expression (8) can be generalized for the robust case as follows:

$$c(v_i) = \frac{\sum_{s \in S} w_i^s}{d(v_i) + 1}. \tag{9}$$

Thus according to our vertex-selecting rule, we give priority to a vertex v_i whose ratio (9) is the greatest (provided that it is not adjacent to other already chosen vertices). The ratio (9) we will be called *relative vertex contribution*, and its numerator will be referred to as *total weight*.

5. Crossover operators

Now we will describe our four original crossover operators that seem to be suitable for solving the RMWIS problem. Any of our crossovers takes as input two existing chromosomes (parents) and produces a new chromosome (child). Thereby both parents correspond to feasible solutions of the considered RMWIS problem instance. On the other hand, the child may for some

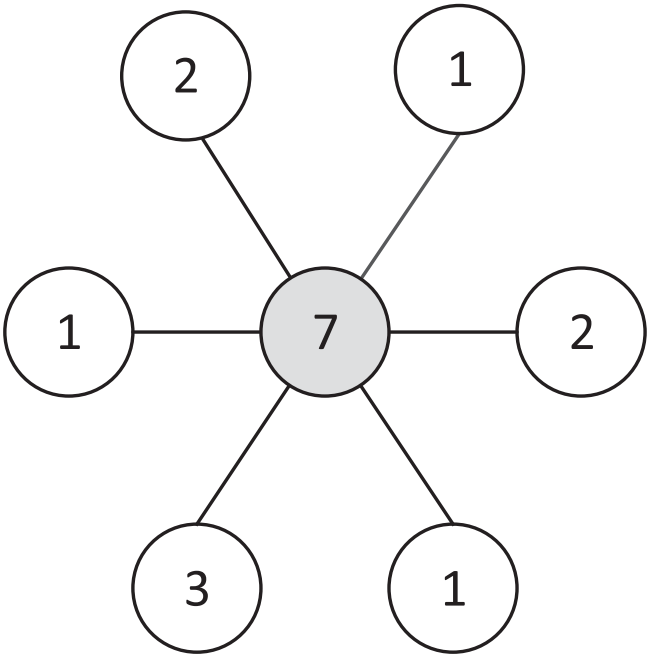

Figure 1. A weighted graph illustrating various vertex-selecting rules.

crossovers turn out to be infeasible, which is then compensated by applying recovery. Each chromosome is represented as a list of chosen vertices, and that list can be sorted in various ways.

- *Alternating vertices crossover (AVX).* Our first crossover operator, called AVX, assumes that within each parent list the vertices are sorted according to their relative vertex contributions (9) in descending order. The child chromosome is constructed by choosing in alternation vertices from the first and from the second parent, thereby following the ordering in both lists. Thus the child is initialized with the first vertex from the first parent, then the first vertex from the second parent is added, then the second vertices from the first and second parent are in turn appended, etc. In case of infeasibility (the candidate vertex is identical or adjacent to some of already chosen vertices) the next vertex from the same parent list is considered. If one of the parent lists gets exhausted, the remaining vertices from the other parent list are directly copied into the child (provided that they are feasible). Although the child constructed in the above way is always feasible, it is still handed over to the recovery operator for possible improvement.
- *Modified alternating vertices crossover (MAVX).* The next operator, denoted with MAVX, is almost the same as AVX. The only difference is the way how the parent lists are initially sorted. In AVX, the parents are sorted according to relative vertex contribution (9) in descending order. In MAVX, they are sorted according to *local* relative vertex contribution, again in descending order. Local relative vertex contribution of a vertex v_i is computed by the same formula (9), except that the degree $d(v_i)$ is now assessed "locally", i.e. by considering only edges connecting v_i with vertices from the other parent.
- *Randomly chosen vertices crossover (RVX).* In RVX, the parent lists do not need to be sorted in any particular order, but it is convenient to assume that they are sorted according to vertex identifiers $1, \ldots, n$. The child chromosome is constructed in n steps. In i-th step it is decided whether to include vertex v_i into the child or not. The decision is guided by the situation found in one of the parents, i.e. if v_i is present in the chosen parent, then it will also be present in the child, and vice-versa. In each step, choosing between the two parents is done randomly, but not with equal probability. More precisely, let the total relative vertex contribution (the sum of relative contributions of included vertices) for the first and for the second parent be t_1 and t_2, respectively. Then the probability of choosing the first parent is $\pi_1 = t_1/(t_1 + t_2)$ and the probability for the second parent is $\pi_2 = 1 - \pi_1$. Thus the parent with larger total relative vertex contribution, being considered as more fit, is more likely to be chosen. Obviously, the child chromosome created by the above procedure may not be feasible, so that it is always sent to the recovery operator for postprocessing.
  It is interesting to note that the RVX operator bears some resemblance to the crossover from [23], and even more resemblance to the crossover from [22]. However, our RVX is more complex and more general since it takes into account multiple scenarios for vertex weights.
- *Modified randomly chosen vertices crossover (MRVX).* The last crossover operator, MRVX, works almost in the same way as RVX. It differs only in the formula for computing the probability π_1. Instead of relative vertex contributions, now the robust objective function (1), (2) and (3), respectively, is taken into account. Indeed, depending on the chosen RMWIS problem variant (max-min, min-max regret, or relative min-max regret), one of the following three formulas is used:

$$\pi_1 = \frac{F_A(\text{first parent})}{F_A(\text{first parent}) + F_A(\text{second parent})}, \quad \text{or}$$

$$\pi_1 = \frac{F_D(\text{second parent})}{F_D(\text{first parent}) + F_D(\text{second parent})}, \quad \text{or}$$

$$\pi_1 = \frac{F_R(\text{second parent})}{F_R(\text{first parent}) + F_R(\text{second parent})}.$$

Note that the max-min variant is a maximization problem, where larger values are regarded as better, while the other two variants are minimization problems, where smaller values are better. Consequently, the above three formulas are adjusted so that, for

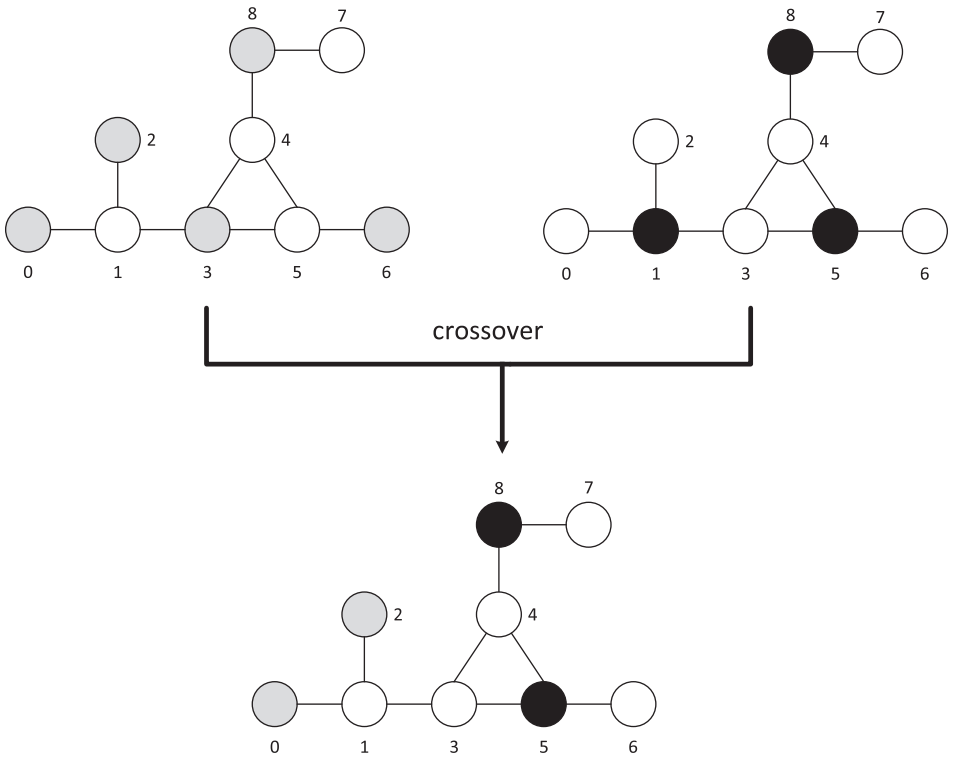

Figure 2. An example illustrating the AVX operator.

each robustness criterion, a more fit parent is more likely to be chosen.

To see more clearly how our crossovers work, we give now two concrete examples. The example presented by Figure 2 corresponds to AVX, while the example from Figure 3 refers to RVX. Within both figures, the same graph is drawn several times in order to show separately each parent, their child, and the recovered version of that child (if recovery is needed). Vertices are labelled (in fact identified) according to their relative vertex contributions (9) sorted in descending order, i.e. vertex 0 has the highest and vertex 8 the lowest contribution.

Let us now analyse in detail the example from Figure 2. The first parent is shown in light grey colour, i.e. it consists of vertices 0, 2, 3, 6 and 8. The second parent is shown in black, i.e. it comprises vertices 1, 5 and 8. We can see that both parents are feasible. The child obtained by AVX combines vertices 0 and 2 from the first parent with vertices 5 and 8 from the second parent, which is indicated by colours. As explained earlier, AVX works in steps. In the first step the child is initialized with the light grey vertex 0. In the second step the black vertex 5 is inserted into the child after skipping the infeasible black vertex 1. In the third step the light grey vertex 2 is appended. In the fourth step the black vertex 8 is chosen. After that, both parent lists become exhausted (since the remaining light grey vertices are infeasible) and the procedure is finished. The obtained child is feasible by construction, but unfortunately it cannot be improved by adding more vertices.

Now we analyse the example from Figure 3. Again, the first and the second parent are shown by light grey and black colour, respectively, and the vertices are labelled according to their relative contribution (9). We see that the first parent contains vertices 0, 2, 3, 6 and 8, while the second parent consists of vertices 1, 4, 6 and 7. The child obtained by RVX in its recovered form comprises vertices 0, 2 and 6 from the first parent, vertex 4 from the second parent, and the reintroduced vertex 7 shown in dark grey. As explained before, RVX works in steps. During its step *i*, RVX decides whether to include vertex *i* into the child chromosome or not. The decision is guided by the situation found in the parent that has randomly been chosen within that step. In our example, the light grey parent is more likely to be chosen since its total relative contribution is larger. Suppose that the random choices made in nine steps are in turn: light grey, light grey, light grey, black, black, black, light grey, light grey and light grey, respectively. Then the initially obtained chromosome turns out to be infeasible since its chosen vertices 4 and 8 are adjacent. The final child chromosome is obtained by applying the recovery operator from the next section. As we can see, the recovery operator first removes the light grey vertex 8 to assure feasibility and then adds the dark grey vertex 7 for improvement.

6. Recovery operator

Our recovery operator takes as input a chromosome that may or may not correspond to a feasible solution of a considered RMWIS problem instance. The operator

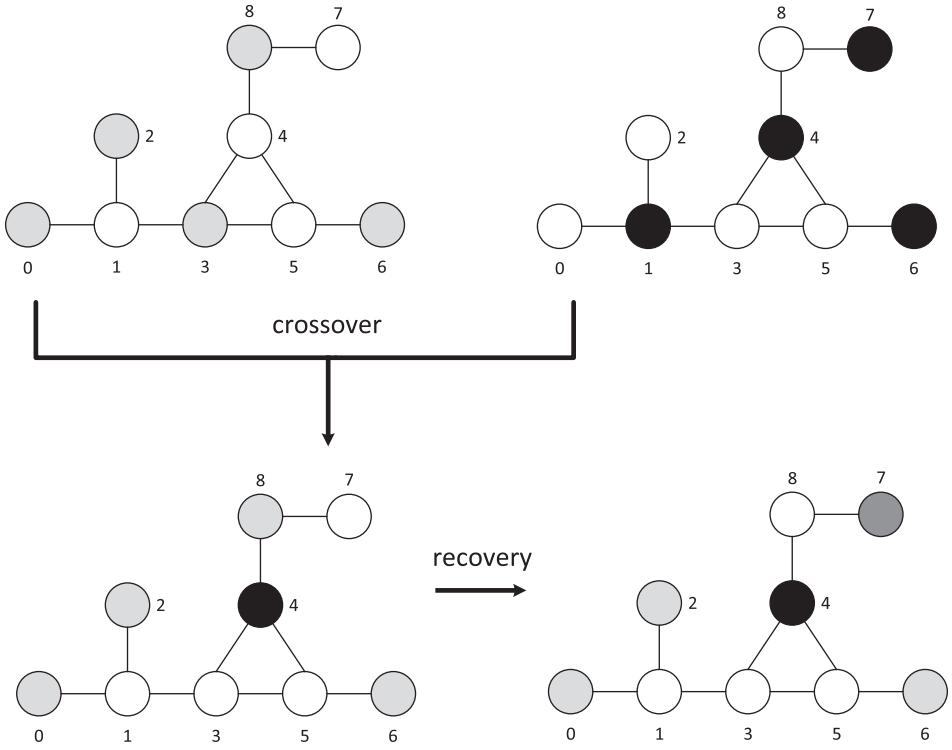

Figure 3. An example illustrating the RVX operator with recovery.

produces a modified version of the same chromosome, which is in the first place feasible, but hopefully also more fit than the original version. The whole computing procedure consists of two phases:

(1) Check if the given chromosome is feasible. If not, modify it so that it becomes feasible.
(2) Try to improve the chromosome obtained in the first phase by further modification.

In the currently implemented variant of the operator, a chromosome is represented as a list of vertices that are sorted according to their total weights in descending order. The first phase of our procedure (assuring feasibility) is accomplished by rewriting the vertices from the original into a new (possibly shorter) list. Vertices are rewritten one by one, by following the original ordering. Thereby, any vertex may be skipped from rewriting if it would spoil feasibility of the already formed part of the new list.

The second phase of computing (improvement) is accomplished by inserting more vertices into the list obtained in the first phase. Thereby, only vertices from the given graph that have not been used in the original chromosome are considered. As we want to maintain genetic diversity, we tend to combine different vertex selecting rules. Therefore, for the second phase of recovery the candidates for insertion are processed in random order. Again, any of them is skipped if its insertion would spoil feasibility of the current list.

A concrete example of recovery has already been shown within Figure 3. Note that in Figure 3 both phases of recovery mentioned above (assuring feasibility, improvement) are applied. An additional example comprising only improvement will be given by Figure 5.

The described recovery operator allows many variants that have not been implemented at this moment, but could easily be tested in the future. For instance, the initial ordering of vertices could be different: instead of total weight we could use some other sorting criterion, e.g. the relative vertex contribution (9). Also, in the improvement phase, the vertices to be inserted can be processed in some other order, e.g. according to their total weights.

7. Mutation operators

Generally speaking, mutation is important because it maintains diversity of chromosomes from one generation to another. It also helps that the whole evolutionary process does not get stuck too early in a local optimum. In case of the RMWIS problem, the role of mutation would be to introduce new vertices, which exist in the given graph but are not used by current chromosomes.

The idea of mutation is that it should take an existing chromosome and modify it through a small and apparently random change. In our case, a chromosome is in fact a list of chosen vertices. A mutation should make a change of that list by removing some vertices and inserting some other vertices. We will describe five mutation operators that work along that line.

- *Simple replacement mutation (SRM).* The SRM operator replaces a randomly chosen vertex in the

original chromosome with a new vertex chosen randomly among those in the graph that have not been used by the original chromosome. The modified chromosome obtained in this way does not need to be feasible, so that it must further be processed by the recovery operator.

It can be noted that the SRM operator is in some aspects similar to the mutation from [23]. Still, our SRM can be regarded as more deterministic, since it always replaces only one vertex within the chromosome.

- *Weight-increasing replacement mutation (WIRM)*. The WIRM operator is similar to SRM. But now the idea is that the overall procedure should be guided or at least influenced by a chosen scenario. Indeed, an old (existing) vertex within the chromosome is again replaced by a new (currently unused) vertex, and feasibility is again reestablished by applying the recovery operator. However, the choice of the two vertices is not completely random. Instead, care is taken that the new vertex has a larger weight than the old one according to the chosen scenario.
- *Weight-decreasing replacement mutation (WDRM)*. The WDRM operator is similar to WIRM. But this time we replace the chosen vertex with a vertex having a smaller weight. Although WDRM spoils solutions, it also introduces new vertices which would otherwise be ignored.
- *Local search replacement mutation (LSRM)*. For each scenario, we make a new better chromosome (if possible) by improving the worst vertex in the original chromosome. The worst vertex v is a vertex with the smallest weight. If there is a neighbour $\bar{v}$ of v that has a larger weight than v and is not adjacent to the other vertices in the original chromosome, we replace v with $\bar{v}$. If there are more such vertices $\bar{v}$, we replace v with the one among them having the largest weight. Finally, among all new chromosomes obtained for different scenarios, we choose the one that achieves the best value of the used robust objective function (i.e. (1), (2) or (3)). Although the constructed chromosome is always feasible, it is still handed over to the recovery operator for possible improvement.
- *Complementary mutation (CM)*. The CM operator can be regarded as an extreme variant of mutation, where a large number of new vertices is introduced simultaneously. For a given original chromosome, CM first determines the set $X \subseteq V$ of vertices in the graph G that are used by that chromosome. Next, the new chromosome is made of vertices from $V \setminus X$ by using the greedy vertex selecting rule according to (9). More precisely, the vertices from $V \setminus X$ are sorted according to their relative vertex contribution in descending order. Then, they are inserted in turn into the new chromosome. Again, any of those insertions is skipped if it would spoil feasibility of the partially formed chromosome. Although the constructed chromosome is always feasible, it is still handed over to the recovery operator for possible improvement.

Some of the described mutation operators are illustrated by the following two figures. Thereby Figure 4 corresponds to SRM and Figure 5 to CM. Within both figures the same graph is again drawn several times to allow showing separately the original chromosome, its mutated version and the recovered version (if recovery is applied). The meaning of vertex labels (identifiers) is the same as in Figures 2 and 3.

Let us proceed with explaining Figure 4 in more detail. The original chromosome consists of vertices 0, 2, 3, 6 and 8, shown in light grey colour on the left-hand side of the figure. The mutated chromosome is shown in a similar manner on the right-hand side, and it comprises vertices 0, 2, 3, 6 and 7. Thereby the original vertices are still light grey, and the newly introduced vertex is black. So we see that SRM has replaced vertex 8 with 7. The obtained chromosome is already feasible, and it cannot further be improved.

Now we analyse Figure 5. The original chromosome is again shown on the left-hand side of the figure - it consists of vertices 0, 2, 3, 6 and 8 coloured light grey. The CM operator constructs the mutated chromosome from the complementary vertices 1, 4, 5 and 7, by using the greedy vertex selecting rule (9) as explained before. The result is shown on the right-hand side of the figure – it comprises vertices 1, 4 and 7 which are painted black. Note that the complementary vertex 5 has been skipped since it is adjacent to 4. The obtained chromosome is feasible, but it is still handed over to the recovery operator for improvement. The recovery operator improves the chromosome by adding vertex 6 painted dark grey in the lower part of the figure.

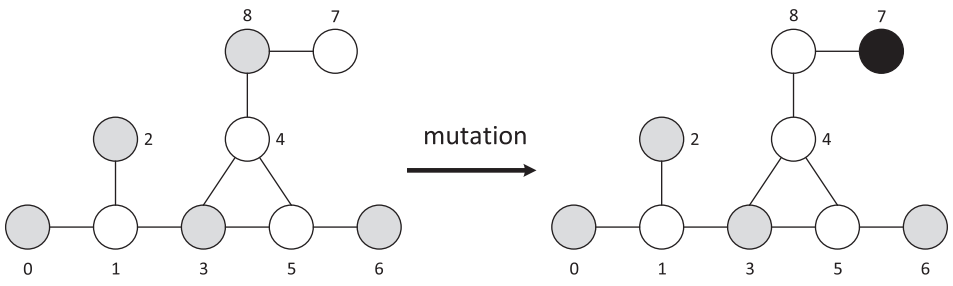

Figure 4. An example illustrating the SRM operator.

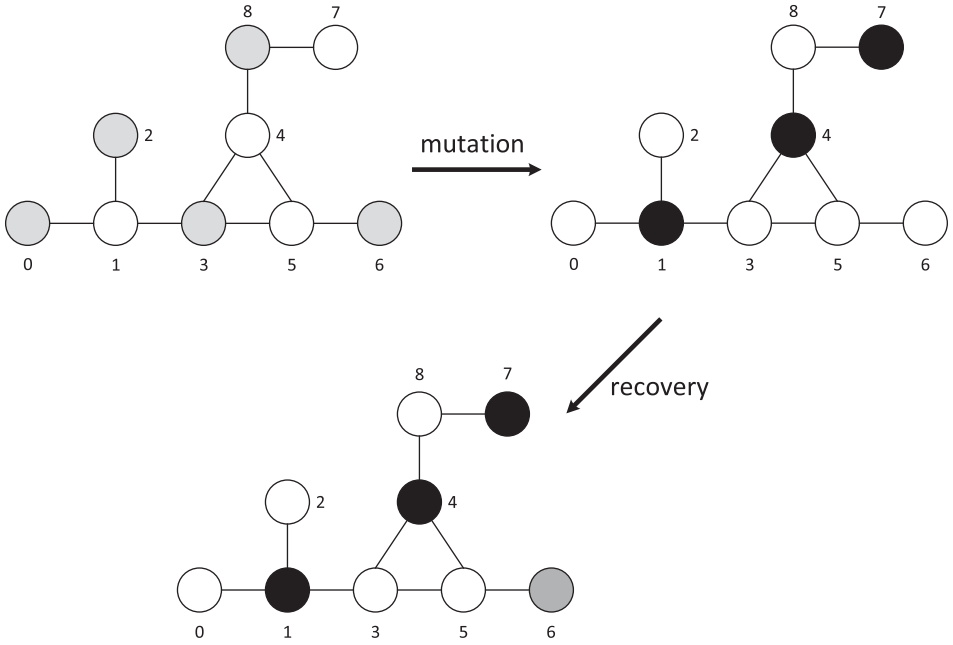

Figure 5. An example illustrating the CM operator with recovery.

8. Testing and results

As there are (to the best of our knowledge) no benchmarks for robust variants of the MWIS problem, we have generated our own two test groups, each comprising 30 problem instances based on random graphs. Any graph from the first group consists of 300 vertices, 30,000 edges and 10 scenarios for vertex weights. A graph from the second group has the same number of vertices and scenarios, but only 20,000 edges. Generally speaking, problem instances from the second group are harder to solve since their graphs are sparser, thus allowing more independent sets. In all instances and all scenarios, vertex weights range from 1 to 300. Indeed, we did not want to have vertices with extremely large weights since they would make optimal solutions too predictable and too easy to find. A full specification of all problem instances is available in our repository at the address `http://hrzz-rodiopt.math.pmf.unizg.hr`.

The chosen problem instances from our two test groups can be regarded as large enough to be non-trivial, but still small enough to be solved exactly by a general-purpose optimization package. Indeed, all instances (written in their ILP form (5), (6) or (7)) can be solved to optimality by the IMB ILOG CPLEX Optimization studio [12]. However, slightly more complex instances, e.g. those with 300 vertices but 10,000 or 15,000 edges, cannot be processed by CPLEX anymore (at least not on our computer) since they produce out-of-memory errors. Moreover, such slightly more complex instances cannot even be solved approximately by relaxation, since the obtained relaxed solutions usually consist of decision variables x_i equal to 0.5, i.e. variables that cannot easily be rounded to a feasible set of 0-s and 1-s.

Our EA for solving the RMWIS problem has been implemented in the C++ programming language by using the Microsoft Visual Studio programming package [31]. The implementation supports all previously mentioned EA variants obtained by combining one among four proposed crossover operators with one among five mutation operators. The program source code can again be downloaded from our repository at the address `http://hrzz-rodiopt.math.pmf.unizg.hr`.

The implemented algorithm has been tested on a computer with an Intel Core i5–6600 K @ 3.50 GHz processor and 16 GB of RAM, running a 64-bit operating system. The same computer has also been used for running CPLEX. During all tests, the number of iterations (generations) in the evolutionary process has been set to 100,000. The frequency of applying crossovers vs. mutations has been set to 95% vs. 5%. It means that in each iteration exactly one crossover or mutation has been executed, while mutation has been activated only occasionally with probability 5%. The initial population has been assembled from three types of solutions of conventional MWIS problem instances. More precisely, for each particular scenario the corresponding (conventional) optimal solution has been included, as well as the greedy solution obtained by using the vertex-selecting rule (8) and the greedy solution obtained by using the rule based on plain weights. If such initial population turned out to be smaller than 30, additional chromosomes have been created from the original ones by applying mutations.

As we have just seen, our EA relies heavily on optimal solutions of conventional problem instances for particular scenarios. Indeed, such solutions are inserted into the initial population. But even more importantly, they are needed as parameters within two of our three robust objective functions, i.e. within (2) and (3). For our relatively small problem instances, the required optimal conventional solutions could be computed exactly by CPLEX. However, for larger instances, exact solving would become impossible due to NP-hardness of the conventional MWIS problem. Therefore, we have decided to perform the required computations only approximately, i.e. by a method that is applicable even for very large problem instances. It means that the so-called exact optimal solutions within the initial population are in fact high-quality approximations of those solutions. Also, the parameters within the robust objective functions used by the EA are in fact approximate.

To find high-quality approximate solutions of the involved conventional problem instances, we have used the same EA as for robust instances. Indeed, a conventional instance can be considered as a robust instance with only one scenario conforming to the absolute criterion of robustness. The initial population in such case is rather small, but as mentioned earlier, it can be increased by mutations. Since our EA partly relies on random choices, repeated executions of the same computational task usually do not produce the same results. Consequently, for any given conventional problem instance, the corresponding computation has been repeated 10 times and the best obtained solution has been retained.

As our EA is an approximate algorithm, the most important indicator of its performance is its accuracy. In our tests involving robust problem instances, we have measured accuracy by computing the relative errors of approximate solutions versus exact (truly optimal) solutions. More precisely, in each test, we have computed relative difference between the robust objective-function value obtained by the EA solution and the corresponding optimal robust objective-function value assessed by CPLEX. Thereby the authentic version of the robust objective function has been used, whose parameters have been determined exactly by CPLEX.

In our tests, we have solved each of the 30 problem instances from each of the 2 test groups by each of the 20 crossover/mutation combinations, according to each of the 3 robustness criteria. As already mentioned, repeated executions of the same task usually do not produce the same solution due to randomized computing. Consequently, for any given problem instance, crossover/mutation combination and robustness criterion, the corresponding computation has been repeated 10 times.

The results of our tests are summarized in Table 1. Six parts of the table correspond to 3 robustness criteria vs. 2 test groups. Each part contains average errors for different combinations of crossovers and mutations. The errors obtained through 10 repeated executions of the same computational task have been averaged. The collected values have further been averaged over test groups. In this way, fairly reliable indicators of performance have been obtained.

Let us now analyse in more detail the results obtained with the absolute robustness criterion. CM can be regarded as the best mutation. For both test groups the CM operator gives the lowest errors combined with the RVX and MRVX crossovers and the lowest in general. If we compare crossover operators, RVX and MRVX are better then the AVX and MAVX for both test groups. Putting it all together, the best average error in general for the first test group is 1.64%, and it is accomplished by RVX combined with CM. On the other hand, the best average error for the second test group is 3.50%, which is attained by MRVX combined with CM.

Now we analyse the results obtained with the robust deviation criterion. Again, CM can be regarded as the best mutation. For both test groups, the CM operator gives the lowest errors combined with the RVX and MRVX crossovers, and the lowest in general. Further, if we compare crossover operators, we see that again RVX and MRVX give much lower errors than the AVX and MAVX, respectively. In general, the best average error for the first test group is 3.60%, and it is again accomplished by RVX combined with CM. Regarding the second test group, the best average error is 7.88%, which is again achieved by MRVX combined with CM.

Finally, we analyse the results obtained with the relative deviation criterion for robustness. For the first test group, it is very noticeable how the RVX crossover produces extremely low errors compared to the other 3 crossovers. For the second test group, RVX is again the best crossover, but it is not so significantly better as for the first test group. When considering all combinations of operators, the lowest possible average errors are produced by the following combinations – RVX combined with WIRM for the first test group and RVX with SRM for the second. The concrete values are 0.25% and 5.96%, respectively. For the first time, CM does not give the lowest relative errors in general.

It is interesting that, for 4 out of 6 cases, the best mutation turns out to be CM. Remember that CM is an extreme variant of mutation, which creates a new chromosome by using vertices not used by the original chromosome, thus bringing more diversity to the population. For the other two cases, CM is not the best, but its error is very close to the lowest.

Note that for all three criteria of robustness and for both test groups our algorithm can assure average relative errors less than 8%. This is quite satisfactory if we take into account that such approximate solutions are obtained much faster than exact solutions. The average execution time of CPLEX and of our EA are

Table 1. Approximation accuracy of our EA – average relative errors.

300 vertices, 30,000 edges, 10 scenarios – absolute robustnes					
	SRM	WIRM	WDRM	LSRM	CM
AVX	5.61 %	7.16 %	6.98 %	7.23 %	6.41 %
MAVX	5.06 %	6.14 %	6.66 %	6.75 %	5.74 %
RVX	2.04 %	2.19 %	2.01 %	2.18 %	1.64 %
MRVX	2.36 %	2.19 %	2.51 %	2.19 %	1.90 %
300 vertices, 20,000 edges, 10 scenarios – absolute robustness					
	SRM	WIRM	WDRM	LSRM	CM
AVX	7.79 %	8.58 %	8.98 %	9.93 %	9.81 %
MAVX	7.5 %	8.89 %	8.97 %	9.02 %	8.66 %
RVX	4.35 %	4.39 %	4.67 %	4.76 %	3.67 %
MRVX	3.83 %	3.53 %	3.82 %	3.78 %	3.50 %
300 vertices, 30,000 edges, 10 scenarios – robust deviation					
	SRM	WIRM	WDRM	LSRM	CM
AVX	10.63 %	12.53 %	12.96 %	12.84 %	12.06 %
MAVX	10.17 %	12.36 %	12.28 %	12.49 %	11.45 %
RVX	4.17 %	4.69 %	4.98 %	5.10 %	3.60 %
MRVX	5.37 %	5.55 %	5.67 %	5.50 %	4.17 %
300 vertices, 20,000 edges, 10 scenarios – robust deviation					
	SRM	WIRM	WDRM	LSRM	CM
AVX	14.07 %	15.95 %	16.66 %	17.10 %	16.16 %
MAVX	13.93 %	14.90 %	15.02 %	15.79 %	14.32 %
RVX	9.60 %	9.93 %	9.76 %	8.82 %	8.50 %
MRVX	8.51 %	8.53 %	9.31 %	8.77 %	7.88 %
300 vertices, 30,000 edges, 10 scenarios – relative robust deviation					
	SRM	WIRM	WDRM	LSRM	CM
AVX	9.05 %	11.03 %	11.73 %	11.44 %	10.71 %
MAVX	9.54 %	10.69 %	11.09 %	10.24 %	9.30 %
RVX	0.34 %	0.25 %	0.36 %	0.37 %	0.39 %
MRVX	4.80 %	4.80 %	4.82 %	5.11 %	4.07 %
300 vertices, 20,000 edges, 10 scenarios – relative robust deviation					
	SRM	WIRM	WDRM	LSRM	CM
AVX	13.66 %	15.75 %	15.77 %	17.27 %	15.98 %
MAVX	13.48 %	15.29 %	15.12 %	15.18 %	14.83 %
RVX	5.96 %	6.49 %	6.14 %	7.87 %	6.21 %
MRVX	7.81 %	7.79 %	8.35 %	8.33 %	6.88 %

Table 2. Average CPU time in seconds needed by CPLEX to find exact solutions.

	Absolute robustness	Robust deviation	Relativerobust deviation
30,000 edges	197	69	75
20,000 edges	978	2064	1942

Table 3. Average CPU time in seconds needed by our EA.

	Absolute robustness	Robust deviation	Relative robust deviation
30,000 edges	2.717	2.780	2.358
20,000 edges	3.136	3.230	2.814

summarized in Tables 2 and 3, respectively. As already stated, CPLEX needs more time for sparser graphs because they allow more independent sets. In such case, our algorithm will produce a solution in around 3 s, which is around 690 times faster than CPLEX would do for robust deviation. The average execution time of our EA depends not only on graph density, but also on the chosen crossover and mutation operators. The values shown in Table 3 correspond to those combinations of operators that assure best accuracy, i.e RVX with CM or WIRM for the first test group and RVX or MRVX with CM or SRM for the second test group.

From Tables 2 and 3, we see that our EA is in some cases several hundreds of times faster than CPLEX. One may wonder how such huge speedup is possible. The explanation is simple: we compare here two algorithms that belong to different categories. Each of them has its own advantages and drawbacks. Indeed, the algorithm used by CPLEX is based on a general branch-and-bound or similar method, which can solve any ILP problem (not only ours). It produces truly optimal solutions, but has very large requirements regarding computing time and space. On the other hand, our EA is customized only to our problem. It runs fast but produces approximate solutions, i.e. high-quality solutions that are not guaranteed to be optimal in the strict mathematical sense.

9. Conclusion

In this paper, we have implemented and tested an approximate algorithm for solving the robust maximum weighted independent set (RMWIS) problem. Our algorithm is based on evolutionary computing, and it relies on custom-designed crossover and mutation operators. We have proposed four crossovers, called AVX, MAVX, RVX and MRVX, together with five mutations, denoted by SRM, WIRM, WDRM, LSRM and CM. We have also considered three robustness criteria: absolute robustness, robust deviation and relative robust deviation.

For testing, we have generated two test groups of RMWIS problem instances. They are large enough to

be nontrivial, but still small enough to be solved exactly by a general-purpose optimization package. The first group contains denser graphs with 300 vertices, 30,000 edges and 10 scenarios for vertex weights. The second group consists of sparser graphs with the same number of vertices and scenarios but only 20,000 edges.

Our tests indicate that performance and relative ranking of particular evolutionary operators are both influenced by robustness criteria and graph density. Indeed, for the first test group, the best solutions are obtained by combining RVX with CM for absolute robustness and robust deviation, but RVX with WIRM for relative robust deviation. Further, for the second group, the best solutions are produced by MRVX combined with CM for absolute robustness and robust deviation, but RVX with SRM for relative robust deviation.

Our algorithm turns out to be between 25 and 690 time faster than the exact algorithm provided by the IBM ILOG CPLEX Optimization studio. The actual speedup depends on the chosen robust criterion and on the involved graph density. The solutions obtained by our algorithm are suboptimal, but their relative errors vs. the corresponding exact optima (measured in terms of robust objective-function values) are quite acceptable. With most efficient combinations of evolutionary operators, the expected errors range between 0.25% and 8%. The actual percentage depends again on the robust criterion and graph density. The operators within our algorithm work well because they successfully combine deterministic greedy techniques for constructing solutions with random processes that maintain diversity of solutions.

The results of this paper are interesting since they provide a practical way of solving large instances of the RMWIS problem. Such large instances can occur in real-life applications. We believe that our approximate algorithm is accurate enough for practical purposes and that its time requirements are tolerable.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This work has been fully supported by Croatian Science Foundation under the project IP-2018-01-5591.

ORCID

Ana Klobučar http://orcid.org/0000-0002-0260-5439
Robert Manger http://orcid.org/0000-0003-0953-6517

References

[1] Aissi H, Bazgan C, Vanderpooten D. Min-max and min-max regret versions of combinatorial optimization problems: A survey. Eur J Oper Res. 2009;197:427–438.
[2] Kouvelis P, Yu G. Robust discrete optimization and its applications. Berlin: Springer; 1997.
[3] Garey MR, Johnson DS. Computers and intractability: a guide to the theory of NP-completness. San Francisco (CA): W.H. Freeman; 1979.
[4] Ben-Tal A, El Ghaoui L, Nemirovski A. Robust optimization. Princeton (NJ): Princeton University Press; 2009.
[5] Bertsimas D, Sim M. Robust discrete optimization and network flows. Math Program. 2003;98:49–71.
[6] Bertsimas D, Sim M. The price of robustness. Oper Res. 2004;52:35–53.
[7] Aissi H, Bazgan C, Vanderpooten D. General approximation schemes for min-max (regret) versions of some (pseudo-)polynomial problems. Discr Optim. 2010;7:136–148.
[8] Bertsimas D, Brown DB, Caramanis C. Theory and applications of robust optimization. SIAM Rev. 2011;53:464–501.
[9] Kasperski A, Zielinski P. Robust discrete optimization under discrete and interval uncertainty: a survey. In: Doumpos M, Zopounidis C, Grigoroudis E, et al., editors. Robustness analysis in decision aiding, optimization, and analytics; p.113–143. Cham: Springer; 2016.
[10] Gross JL, Yellen J, Zhang P. Handbook of graph theory. 2nd ed. Boca Raton (FL): CRC Press; 2014.
[11] Jungnickel D. Graphs, networks and algorithms. 4th ed. Berlin: Springer; 2013.
[12] IBM ILOG CPLEX Optimization Studio. CPLEX User's Manual, Version 12, Release 8. IBM Corporation [cited 2018 Oct 10]. Available from: https://www.ibm.com/support/knowledgecenter/SSSA5P_12.8.0..
[13] Kasperski A, Zielinski P. Complexity of the robust weighted independent set problems on interval graphs. Optim Lett. 2015;9:427–436.
[14] Klobučar A, Manger R. Independent sets and vertex covers considered within the context of robust optimization. Math Commun. 2020;25:67–86.
[15] Klobučar A, Manger R. Solving robust variants of the maximum weighted independent set problem on trees. MDPI Math. 2020;8. [cited 2020 Mar 12]. Available from: https://doi.org/10.3390/math8020285.
[16] Talla Nobibon T, Leus R. Robust maximum weighted independent-set problems on interval graphs. Optim Lett. 2014;8:227–235.
[17] Lamm S, Schulz C, Strash D, et al. Exactly solving the maximum weight independent set problem on large real-world graphs. In: Kobourov S, Mayerhenke H, editors. 2019 Proceedings of the Meeting on Algorithm Engineering and Experiments (ALENEX), San Diego, CA, Jan 7–8, 2019; p.144–158. Philadelphia (PA): SIAM; 2019.
[18] Kako A, Ono T, Hirata T, et al. Approximation algorithms for the weighted independent set problem in sparse graphs. Discr Appl Math. 2009;157:617–626.
[19] Sakai S, Togasaki M, Yamazaki K. A note on greedy algorithms for the maximum weighted independent set problem. Discr Appl Math. 2003;126:313–322.
[20] Cai S, Hou W, Lin J, et al. Improving local search for minimum weight vertex cover by dynamic strategies. In: Lang J, editor. Proceedings of the 27-th International Joint Conference on Artificial Intelligence (IJCAI-18), Stockholm, Sweden, Jul 13-19, 2018; p.1412–1418. Freiburg-Vienna: IJCAI; 2018.
[21] Nogueira B, Pinheiro RGS, Subramanian A. A hybrid iterated local search heuristic for the maximum weight independent set problem. Optim Lett. 2018;12:567–583.

[22] Hifi M. A genetic algorithm-based heuristic for solving the weighted maximum independent set and some equivalent problems. J Oper Res Soc. 1997;48:612–622.
[23] Nayeem SMA, Pal M. Genetic algorithmic approach to find the maximum weight independent set of a graph. J Appl Math Comput. 2007;25:217–229.
[24] Korte B, Vygen J. Combinatorial optimization – theory and algorithms. 5th ed. Berlin: Springer; 2012.
[25] Saha A, Pal M, Pal TK. Selection of programme slots of television channels for giving advertisement: a graph theoretic approach. Inf Sci (NY). 2007;177:2480–2492.
[26] Barth L, Niedermann B, Nöllenburg M, et al. Temporal map labeling: a new unified framework with experiments. In: Ali M, Newsam S, Ravada S, et al., editors. Proceedings of the 24-th ACM SIGSPATIAL International Conference on Advances in GeographicInformation Systems (SIGSPATIAL 2016), San Francisco Bay Area, CA, Oct 31 – Nov 3, 2016; Article 23, p.1–10. New York (NY): ACM; 2016.
[27] Eiben AE, Smith JE. Introduction to evolutionary computing. 2nd ed. Berlin: Springer; 2015. (Natural Computing Series).
[28] Michalewicz Z. Genetic algorithms + data structures = evolution programs. 3rd ed. New York (NY): Springer; 1996.
[29] Talbi EG. Metaheuristics – from design to implementation. Hoboken (NJ): Wiley; 2009.
[30] Puljić K, Manger R. Comparison of eight evolutionary crossover operators for the vehicle routing problem. Math Commun. 2013;18:359–375.
[31] Microsoft Corporation. Visual studio documentation 2017. [cited 2018 Oct 10]. Available from: https://docs.microsoft.com/en-us/visualstudio/?view=vs-2017.