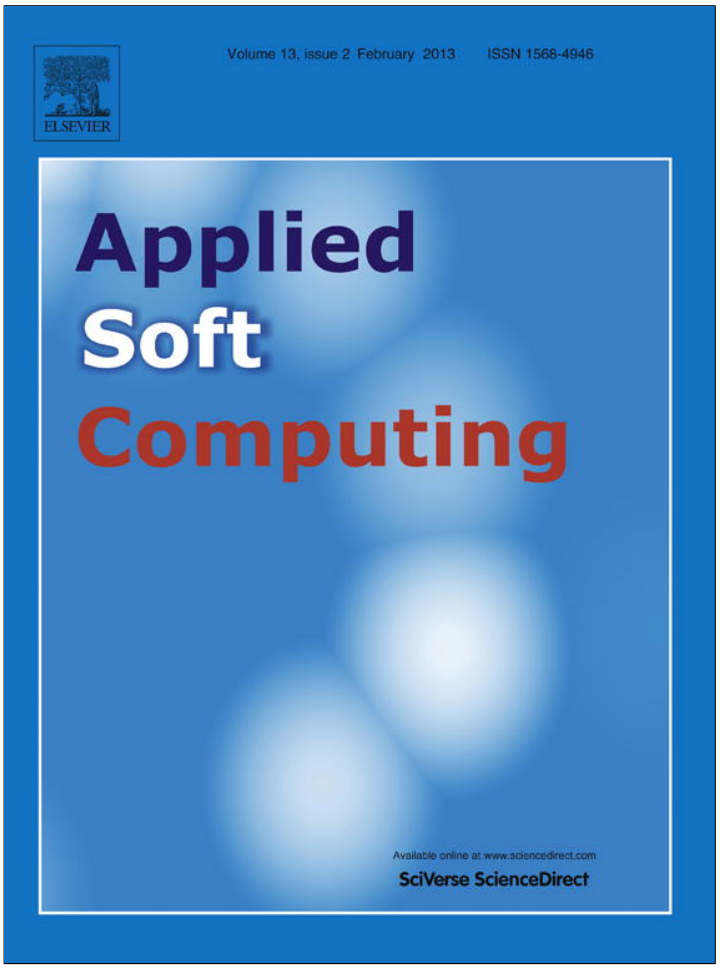

Applied Soft Computing 13 (2013) 1303–1313

Contents lists available at SciVerse ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

An electromagnetism metaheuristic for solving the Maximum Betweenness Problem

Vladimir Filipović [a,*], Aleksandar Kartelj [a], Dragan Matić [b]

[a] University of Belgrade, Faculty of Mathematics, Studentski trg 16, 11000 Belgrade, Serbia
[b] University of Banjaluka, Faculty of Mathematics and Natural Sciences, Mladena Stojanovića 2, 78000 Banjaluka, Bosnia and Herzegovina

ARTICLE INFO

Article history:
Received 28 November 2011
Received in revised form
30 September 2012
Accepted 22 October 2012
Available online 28 November 2012

Keywords:
Betweenness problem
Electromagnetism-like mechanism
Combinatorial optimization

ABSTRACT

In this paper we present an electromagnetism (EM) metaheuristic for solving NP hard Maximum Betweenness Problem (MBP). A new encoding scheme with appropriate objective functions is implemented. Specific representation of the individuals enables the EM operators to explore the searching space in a way that achieves high quality solutions. An effective 1-swap based local search procedure improved by the specific caching technique is performed on each EM point. The algorithm is tested both on real and artificial instances from the literature. Experimental results show that the proposed EM approach achieves all previously known optimal solutions, except one, and achieves the best-known solutions or outperforms other approaches on all large-scale instances, except two. Provided statistical analysis indicates that the EM approach is significantly better than other approaches.

1. Introduction

The betweenness problem (BP) is a well known combinatorial optimization problem. For a given finite set S of n objects $S = \{x_1, x_2, \ldots, x_n\}$ and a given set C of triples $(x_i, x_j, x_k) \in S \times S \times S$, the BP is a problem of determination of the total ordering of the elements from S, such that triples from C satisfy the "betweennesses constraint", i.e. the element x_j is between the elements x_i and x_k. The problem presented in this paper, called Maximum Betweenness Problem (MBP), deals with finding of total ordering that maximizes the number of satisfied constraints.

Since the set S is finite, each element of S can be indexed by $i = 1, \ldots, n$, where $n = |S|$. This 1-1 correspondence between the elements from S and the set $\{1, 2, \ldots, n\}$ introduces a linear ordering of the elements from S: if $x_i, x_j \in S$, then x_i "is before" x_j if and only if $i < j$. So, without the loss of generality, we can suppose that the starting set S is the set $\{1, 2, \ldots, n\}$. Now, it is clear that for solving the MBP we can consider the 1-1 functions $f: \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$. Any 1-1 function from a finite set to the same finite set is actually one permutation of that finite set and vice versa, each permutation can be considered as a 1-1 function from the finite set onto itself. In conclusion, MBP is finding the permutation π of S that maximizes the number of triples (a_i, b_i, c_i), such that $\pi(a_i) < \pi(b_i) < \pi(c_i)$ or $\pi(c_i) < \pi(b_i) < \pi(a_i)$.

Example 1. Suppose that $n = 5, S = \{1, 2, 3, 4, 5\}$ and the collection C contains 6 triples: $(1, 5, 2), (3, 4, 2), (4, 1, 5), (2, 1, 4), (5, 4, 3)$ and $(1, 4, 3)$. The optimal solution is the permutation $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 5 & 1 & 2 & 4 \end{pmatrix}$. The objective value is 6 which means that all constraints are satisfied. It can easily be seen that the triples are respectively mapped to $(3, 4, 5), (1, 2, 5), (2, 3, 4), (5, 3, 2), (4, 2, 1)$ and $(3, 2, 1)$.

The decision version of the betweenness problem is to decide if all constraints can be simultaneously satisfied by a total ordering of the elements from S. The decision version of the problem is proved to be NP hard in [1].

The MBP and other variants of betweenness problems belong to a class of discrete optimization problems which have a lot of applications in various fields of science. For example, the MBP is used for solving some physical mapping problems in molecular biology [2]. During the radiation hybrid experiments, the X-rays are used to fragment the chromosome. If the markers on the chromosome are more distant from one another, the probability that the given dose of an X-ray will break the chromosome between them is greater. In that way markers are placed on two separate chromosomal fragments. By estimating the frequency of the breaking points, and thus the distances between markers, it is possible to determine their order in a manner analogous to meiotic mapping. In this context, improvement of the radiation experiment is achieved by finding the total ordering of the markers that maximizes the number of satisfied constraints. The software package RHMAPPER ([3,4]) uses this approach to produce the order of framework markers, employing two greedy algorithms for solving the betweenesses problem.

* Corresponding author. Tel.: +381 11 202 78 01; fax: +381 11 202 78 01.
E-mail addresses: vladaf@matf.bg.ac.rs (V. Filipović), aleksandar.kartelj@gmail.com (A. Kartelj), matic.dragan@gmail.com (D. Matić).

1568-4946/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.asoc.2012.10.015

Detailed analysis about the experiments of radiation hybridization is beyond the scope of this paper and can be investigated in [5–7].

The number of violations, i.e. the number of triples for which the betweenness constraint is not satisfied can be penalized by weights, and the problem which deals with the minimization of the total sum of these weights is called Weighted Betweenness Problem (WBP). This problem appears in computational biology [8], more precisely, in the physical mapping with the end probes. For each element i, from a given set of m "clones", two, so-called, end probes, $t(i)$ and $h(i)$, are associated. For each pair of a clone i and a probe $j \in \{1, 2, \ldots, n\} \setminus \{t(i), h(i)\}$ either the betweenesses condition $(t(i), j, h(i))$ is satisfied or not. If each non-satisfied constraint is penalized, then the WBP is to find a linear order of probes which minimizes the total sum of all penalties.

The rest of the paper is organized as follows. The next section gives a literature review of the MBP. The mixed integer linear programming model (MILP) presented in [9] is provided in Section 3. Electromagnetism approach is presented in Section 4 and computational results which prove the usability of the proposed EM approach are provided in Section 5. The last section holds the conclusion of our research.

2. Literature review

The first paper which deals with the betweenness problem dates back to the late 1970s, when Opatrny [1] showed that the problem of deciding whether the n points can be totally ordered, while satisfying the m betweenness constraints, is NP-complete. The analysis of the problem complexity was deeper studied in [2], where the authors proved that the problem is MAX SNP complete [10]. They also showed that for every $\alpha > 47/48$, finding the total order that satisfies at least αm betweenesses constraints is NP hard, even if all the constraints are satisfiable. It is easy to conclude that the probability that a given constraint (i, j, k) is satisfied by a randomly chosen order of elements is 1/3, because among a total of six permutations of the triple (i, j, k), two of them place j in the middle. So, by randomly choosing the ordering, we can expect that about 1/3 of all constraints will be satisfied. A polynomial time algorithm for solving a similar problem of determining if there is no feasible solution or finding a total order that satisfies at least 1/2 of the m constraints is given in [2]. The algorithm transforms the problem into a set of quadratic inequalities and solves a semidefinite programming (SDP) relaxation of them in $\mathbf{R}^n$. If the vectors $v_1, v_2, \ldots, v_n \in \mathbf{R}^n$ are a feasible solution of the SDP, then $v_i, i = 1, \ldots, n$ corresponds to one element i from the starting set S. These n solution points are then projected on a random line through the origin. The guaranteed performance is shown by using simple geometric properties of the semidefinite programming solution. A comprehensive study of using semidefinite programming technique can be found in [11].

Several variations on the ordering of the elements of a given set are presented in [12]. Depending on the admitted constraints, the authors provided an efficient algorithm, or proved the NP-completeness. One custom model, which introduces new local constraints, arises from an object-oriented model of an information system, where two classes (L and M) are connected in directed association, and the third association class (K) details the relationship between the first two classes. In order to preserve information about the established associations in the association table, a certain memory is needed. It should be organized in a way that one of the three objects that participate in a link would make the entry into that association table. Since all three identifiers are needed for this, the only object of each link capable to do this is the one stored last. Moreover, because of the restricted visibility in the model, this must not be the object of class M, for it cannot access the other identifiers. To summarize, for each triple (i, j, k) of objects from classes (K, L, M) that constitute such a link, i or j must be stored after k. This is the reason for the aforementioned requirement $k < i$ or $k < j$ for the total order. For this custom model, the authors present an efficient algorithm which is a generalization of topological sorting.

Another theoretical result in the field of the betweenness problem is studied in [13]. The author considers the variant of the betweenness problem where all constraints should be simultaneously satisfied by a total ordering. Based on the binary relaxation of the betweenness constraint and some other concepts, the author proves that, if the instance is chosen randomly, the betweenness constraints are satisfiable with high probability if the number of randomly picked clauses is $\leq 0.92 \cdot n$, where n is the number of variables considered.

In [8], the authors analyzed the variant of betweeness problem where non-satisfied constraints are penalized by weights (this is the WBP problem), trying to minimize the total sum of all penalties. They proposed the integer programming (IP) formulation of the WBP based on the IP formulation of the weighted consecutive ones problem. In addition, the authors developed a new branch-and-cut algorithm for solving the WBP.

Recent notable results in solving the MBP are presented in [14,9,15]. In [14], a genetic algorithm (GA) for solving the MBP is applied. The GA uses specific integer encoding scheme and standard genetic operators: one-point crossover, standard mutation and fine-grained tournament selection. The GA was successfully tested on random generated instances, achieving the optimal solutions for smaller instances, and good quality results for larger ones. In [15], the proposed GA is improved by the hybridization with a local search technique. In each step of the GA execution, the local search chooses two indices i and j and swaps the values $p(i)$ and $p(j)$. If the improvement is obtained, the current, locally-created, permutation is pronounced as the best one, otherwise, another pair of indices (i, j) is chosen. The proposed approach gives better or equal results on all instances compared to the GA without the local search. In our paper, we deal with the same problem as in [14,15], using EM approach instead of GA.

A Mixed Integer Linear Programming model (MILP) for solving the MBP is introduced in [9]. The paper presents the MILP formulation, as well as the proof of the validity of the formulation. The experiments were performed on the same instances used in [14], and the results obtained by the CPLEX 10.1 solver [16] were compared to the results obtained by total enumeration technique. The MILP formulation from [9] is presented in the next section.

3. Mathematical formulation

Let n be a number of elements in a finite set S. As it has already been mentioned, without the loss of the generality, we can assume $S = \{1, 2, \ldots, n\}$. Let C be a collection of m triples from $S \times S \times S$ and let the i-th triple from the collection C, $i = 1, \ldots, m$ be denoted as (a_i, b_i, c_i). Also, let α be a real number from $(0, 1]$.

Suppose the 1-1 function f is known in advance, $f: \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$. Four sets of variables are introduced: The variables x are defined as

$$x_j = \frac{f(j) - 1}{n}, \quad j = 1, \ldots, n \tag{1}$$

For each $i = 1, \ldots, m$, the binary variables y, z and u are defined as

$$y_i = \begin{cases} 1, & f(a_i) < f(b_i) < f(c_i) \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

$$z_i = \begin{cases} 1, & f(a_i) > f(b_i) > f(c_i) \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

$$u_i = y_i \vee z_i = \begin{cases} 1, & f(a_i) > f(b_i) > f(c_i) \vee f(a_i) < f(b_i) < f(c_i) \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

Because of binary nature of the variables u_i, $i = 1, \ldots, m$, the variables y_i and z_i cannot simultaneously have value 1. Now, the MILP model is formulated as follows:

$$\max \sum_{i=1}^{m} u_i \quad (5)$$

subject to

$$u_i = y_i + z_i, \quad i = 1, \ldots, m \quad (6)$$

$$x_{a_i} - x_{b_i} + y_i \leq 1 - \frac{\alpha}{n}, \quad i = 1, \ldots, m \quad (7)$$

$$x_{b_i} - x_{c_i} + y_i \leq 1 - \frac{\alpha}{n}, \quad i = 1, \ldots, m \quad (8)$$

$$-x_{a_i} + x_{b_i} + z_i \leq 1 - \frac{\alpha}{n}, \quad i = 1, \ldots, m \quad (9)$$

$$-x_{b_i} + x_{c_i} + z_i \leq 1 - \frac{\alpha}{n}, \quad i = 1, \ldots, m \quad (10)$$

$$x_j \in [0, 1], \quad j = 1, \ldots, n \quad (11)$$

$$y_j, z_j, u_j \in \{0, 1\}, \quad j = 1, \ldots, m \quad (12)$$

As can be seen, there are n real variables (variables x) and $3m$ binary variables (variables y, z and u). There are $5m$ constraints. The parameter α is introduced in order to make α/n greater than a round-off error, so α has to be greater than 0. An upper bound on α is 1, in order to ensure that the system (6)–(12) always has a nonempty solution set. Now, the $Obj_{MILP}(x; y; z; u) = \sum_{i=1}^{m} u_i$ subject to (6)–(12). A detailed proof of the validity of the MILP formulation is provided in [9].

4. EM method for solving MBP

In recent years, many combinatorial optimization problems became a focus of scientific interest. Due to their theoretical and practical importance, an impressive number of algorithms have been developed for solving them. A large class of these algorithms is referred to as the hybrid metaheuristics that combine various algorithmic approaches. Often, hybrid metaheuristics combine a certain global strategy with a local search, which iteratively tries to change the current solution to a better one, placed in some neighborhood of the current solution. Detailed description of the aforementioned combinatorial optimization techniques is beyond the paper's scope and can be found in [17,18].

The electromagnetism method (EM) is a metaheuristic algorithm which is introduced by Birbil and Fang in [19]. EM utilizes an attraction-repulsion mechanism to move sample points towards optimality. Each point (particle) is treated as a solution and a charge is assigned to each particle. Better solutions possess stronger charges and each point has an impact on others through charge. The exact value of the impact is given by Coulomb's Law. This means that the power of the connection between two points will be proportional to the product of charges and reciprocal to the distance between them. In other words, the points with a higher charge will force the movement of other points in their direction more strongly. Beside that, the best EM point will stay unchanged. The charge of each point relates to the objective function value, which is the subject of optimization. A comprehensive study which analyzes the convergence of the EM approach is beyond the paper's scope and can be found in [20].

Recently, a number of continuous and discrete optimization problems have been solved by the EM approach. Some recent successful applications of the EM are used for solving the following problems:

- the unicost set covering problem [21];
- the uncapacitated multiple allocation hub location problem [22];
- automatic detection of circular shapes embedded into cluttered and noisy images [23];
- feature selection problem [24].

```
input data: maxIter, EMPointsNo, maxRep;
setControlParameters(maxIter, EMPointsNo, maxRep);
y= createEMPoints(EMPointsNo);
while(iteration<maxIter) do
    for i=1 to EMPointsNo do
        calculateObjectiveValue(y_i); // fitness evaluation
        localSearch(y_i); // partial fitness evaluation
    endfor
    calculateChargesAndForces(y);
    move();
    if (solutionUnchangedFor(maxRep))
        break;
endwhile
solutionPrint();
```

Fig. 1. EM pseudocode.

The proposed EM algorithm for solving the MBP is given by the pseudocode shown in Fig. 1. In the initialization phase, the control parameters are specified: maximal number of iterations, the number of EM points, and maximal number of repetition of the same solution. After the input and initialization procedures, the algorithm enters into the main loop. In each iteration of the main loop, for each EM point y_i, the objective function and local search procedures are called. In Fig. 1 we stressed that during the execution of the objective function, the fitness evaluation is performed. Also, a partial fitness evaluation is done during the local search procedure, which is explained in Section 4.3. After the inner loop finishes, the objective function values of all EM points are used to calculate the values of the charges and forces. According to these values, the EM points are moved towards the solution space. The iteration process ends when either the maximal number of iteration is reached, or the same best solution is not changed for the predefined maximal number of iterations.

In the rest of this section, we describe, in detail, the adopted EM for solving the MBP.

4.1. Unit representation and the objective function

In order to maintain the search effectiveness of the algorithm, choosing an appropriate representation of the candidate solution plays a key role. In the case of MBP, each EM point in the solution set (set of all EM points) is related to one ordering of the set $S = \{1, 2, \ldots, n\}$. That ordering is used for determining the number of satisfied constraints in the objective function. Let us represent the EM point as a n-dimensional vector of real valued coordinates, taking values from [0, 1] and let us denote that vector as $x = (x_1, x_2, \ldots, x_n)$, $x_i \in [0, 1]$, $i = 1, \ldots, n$. For a given EM point x, each element of the set S corresponds to one coordinate of that point and vice versa. The point x determines the corresponding ordering relation: if i and j are two elements from S, then $i \prec j \Leftrightarrow x_i < x_j$. This fact introduces the objective function which calculates the total number of

satisfied constraints from the set $C = \{(a_l, b_l, c_l) | l = 1, 2, \ldots, m\}$, by the expression (13).

$$obj(x) = |\{(a_l, b_l, c_l) : (a_l, b_l, c_l) \in C, x_{a_l} < x_{b_l} < x_{c_l} \text{ or } x_{c_l} < x_{b_l} < x_{a_l}\}|. \quad (13)$$

If all coordinates x_i are different, then the ordering proposed by the EM point x induces a 1-1 function $f: S \to S$, which is actually a permutation of the set S. The function f is determined by sorting the set of indices of the point x by the criteria proposed by the ordering relation: if i and j are two indices, then $i \prec j \Leftrightarrow x_i < x_j$. If i_j, $j = 1, \ldots, n$ denotes the index of the element j in the sorted array, then the permutation f is defined as $f(j) = i_j$. Note that for calculating the objective function of the ordering (13) we do not need to determine the permutation.

Example 2. If $n = 4$ and vector $x = (0.98, 0.86, 0.37, 0.78)$, then the corresponding ordering is $3 \prec 4 \prec 2 \prec 1$. The corresponding permutation is $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 1 & 2 \end{pmatrix}$.

We should note that, during the execution of the algorithm, the situations $x_i = x_j$ for some i, j can appear. This implies that, for the corresponding function f, neither $i \prec j$, nor $j \prec i$ is satisfied for the elements $i, j \in S$. In the Lemma 3 from [9], the authors proved that for each such ordering (function f) there exists a 1-1 function g, $g: S \to S$ such that obj(g) ≥ obj(f), where obj(g) and obj(f) are the objective values for g and f. The proof is based on the fact that all the constraints containing both i and j are not satisfied for sure. So, the objective function applied to this ordering returns less, or, in best case, equal value, than in the case when $x_i \neq x_j$, with holding the ordering of the rest of the elements unchanged. As a conclusion, we could say that these situations can appear, but the optimization process, led by the objective function, will discard them.

Motivation of this approach of encoding is justified by the following reasoning: during the execution of the EM algorithm, the points are continuously moved from one position to another, depending on the calculated forces. Due to the fact that there are "more" points in continuous space than in a discrete one, one ordering will not be transformed into another by each such movement. Further, minor movements of the points should not change the objective values. This means that, if one point x proposes one ordering, then each vector from some neighborhood of the point x should be related to the same ordering. The proposed encoding satisfies this requirement, because, if $x = (x_1, x_2, \ldots, x_n)$ is a vector (point) in continuous space, then each vector $x' = (x'_1, x'_2, \ldots, x'_n)$ satisfying the condition $x'_i < x'_j \leftrightarrows x_i < x_j$ for all $i, j \in \{1, 2, \ldots, n\}$ is related to the same ordering and, also, the same objective value. Let us illustrate this approach in Fig. 2, where only the 2-dimensional restriction of the whole space is shown. Without the loss of generality, suppose that the vector $x = (x_1, x_2, \ldots, x_n)$ is already sorted and corresponds to the appropriate ordering. The marked area (triangle) denotes the set of points (x'_i, x'_{i+1}), for which the condition $x'_i < x'_{i+1}$ is satisfied. Combining the pair (x'_i, x'_{i+1}) with the remaining coordinates of the vector x, the formed vector x' determines the same ordering, related to the same objective values. In other words, as it is illustrated in Fig. 2, any movement of the pair (x_i, x_{i+1}) inside the marked triangle, will not change the objective value.

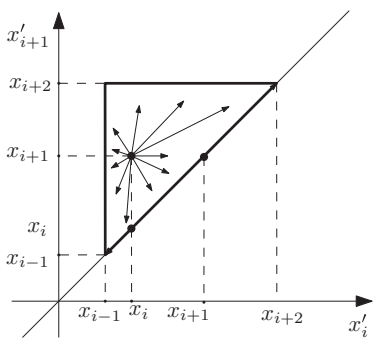

Fig. 2. 2-Dimensional restriction of the searching space.

```
input data: y_k // k-th EM point
p = decode(y_k);
satCache=prepareCache();
improvement=true;
while(improvement)
    improvement=false;
    foreach(unordered pair of indices {i,j})
      if(improvement)
          break;
      df=inversionPayoff(i,j,p,satCache);
      if(df>0)
          applyInversion(i,j,p);
          reevaluateCache(i,j);
          fval=fval+df;
          improvement = true;
      else
          discardInversion(i,j,p);
          //no reevaluation of cache needed
    endforeach
endwhile
```

partial fitness evaluation

Fig. 3. Improved local search pseudocode.

This neighborhood is successively spread by introducing the neighborhoods of other pairs of successive coordinates, and the EM points from those neighborhoods determine the same objective function related to the starting point. This approach guarantees that the whole continuous space is covered by the searching process and that the neighboring points correspond to the same objective functions.

4.2. Initialization phase

After the data input, during the initialization phase, *EMPointsNo* points are randomly generated: for each point, the vector x is randomly formed by choosing the coordinates x_i from [0, 1] at random. According to the mapping analyzed in the previous section, *EMPointsNo* random orderings are created. The quality of the particular EM point is measured by calculating its objective value by the formula (13).

4.3. Local search

In each iteration, the algorithm is trying to improve each EM point. This is done in the special local search procedure called improved LS, which combines the 1-swap local search approach and caching technique. Pseudocode for the improved local search is provided in Fig. 3. Firstly, Based on the current point y_k, a permutation p is determined. In the sub-procedure called prepareCache, a cached structure *satCache* is created: the ith element of this list represents the number of satisfied constraints in which the ith element occurs. The main purpose of the structure *satCache* is to speed up the

```
input data: i,j,p; // i, j - indices that are inverted, p - current permutation
df=0;
df=df-satCache[i] ;
df=df-satCache[j];
swapElements(p[i],p[j]);
ci=constraintsWithElement(i);
for k=1 to ci.length do
        if(ci.length-k<abs(df))
            return -1;
        // ci.l, ci.m, ci.r - left, middle and right element in constraint ci
        l=p[ci.l]; m=p[ci.m]; r=p[ci.r];
        if(l<m<r || l>m>r)
            df++;
endfor
cj=constraintsWithElement(j);
for k = 1 to cj.length do
        if(cj.length-k<abs(df))
            return -1;
        // cj.l, cj.m, cj.r - left, middle and right element in constraint cj
        l=p[cj.l]; m=p[cj.m]; r=p[cj.r];
        if(l<m<r || l>m>r)
            if(cj.l==i || cj.m==i || cj.r==i)
                if(satisfiedBeforeSwap(cj))
                    df++;
            else
                df++;
endfor
return df;
```

Fig. 4. Pseudocode for inversionPayoff procedure.

calculation of the potential benefit of the swap. The caching effect is reached because the number of satisfied constraints for each element is calculated only once (in the procedure prepareCache), and the update of the structure is performed only to the indices figuring in the swap, while the rest of the structure is unchanged.

After the procedure prepareCache finishes, the main loop tries to improve the solution until no improvement is found. In the inner loop, each pair of elements is swapped, and then the partial evaluation of objective value (which is in our case same as the fitness value) is performed. In order to calculate the difference between the objective values before and after the swap, the sub-procedure inversionPayoff(*i*, *j*, *p*, *satCache*) is called. The inner loop finishes when the first improvement occurs. The pseudocode of the procedure inversionPayoff is given in Fig. 4.

The output of the procedure inversionPayoff is the value called *df*, which indicates the difference between the number of constraints, which are now satisfied, and the number of constraints violated by the swap.

Initially, *df* is set to zero. After that, the numbers of violated constraints, where *i*th and *j*th element appear, are subtracted from *df*. These numbers are maintained in the structure satCache which reduces the need for their frequent evaluation. The next step is to perform the swap of elements in positions *i* and *j*, and then to increase the value of *df* according to the number of newly satisfied constraints that correspond to the new positions of the *i*th and *j*th element. Firstly, this is done for the constraints where the element *i* appears. However, when considering the index *j*, a problem could arise, because *i*th and *j*th element could appear in the same constraint. If this is the case, it is checked whether the analyzed constraint was satisfied before, and if so, for that constraint *df* was initially decreased (by one) two times, by both satCache[*i*] and satCache[*j*]. Therefore, the value of *df* needs to be increased by one.

Additionally, a certain speed-up can be achieved by evaluating the condition, if the number of remaining (not analyzed) constraints is less than *abs*(*df*). If so, *df* cannot become greater than zero, and in this case, the iteration process stops and the procedure returns the value $df = -1$.

In the main procedure improved LS the value *df* is examined, and if $df > 0$ the improvement is achieved, the *satCache* structure is updated in the procedure reevaluateCache(*i*,*j*). After that, the objective value is increased by *df* and the local search restarts. Otherwise, another pair $\{i, j\}$ is chosen to be swapped. If improvement is never done for the current EM point, local search stops after all the pairs $\{i, j\}$ are analyzed.

Simple theoretical consideration and experiments prove that improved LS is faster than the local search procedure described in [15]. In [15], local search is based on the same 1-swap approach. The problem with this approach is that it deals with the list of satisfied constraints and in each iteration of local search, for given *i* and *j*, the list is updated twice. Firstly, all satisfied constraints in which *i* and *j* figure are removed, and after the swap, new satisfied constraints are added. In our approach, improved LS deals with list of different nature, holding only the information of the total number

of satisfied constraints. This approach enables the list to be updated only once per iteration.

4.4. The relocation of the EM points

During the calculation of charges, potential solutions (EM points) are being evaluated according to the following formula:

$$q_i = \exp\left(-n\frac{f(x_{best}) - f(x^i)}{\sum_{k=1}^{m} f(x_{best}) - f(x^k)}\right) \tag{14}$$

The charge of the point is calculated according to the relative efficiency of the objective function values in the current set of EM points. In the formula (14), n is a dimension space and x_{best} is the best solution in the solution space after the local search. Note that, if all the objective values of all particles are equal, the nominator of the fraction in the expression (14) would be 0, and in that case, the total charge of each particle is set to the default value 1.

The total force, F_i, exerted on candidate solution x_i is calculated by the formula (15):

$$F_i = \begin{cases} \displaystyle\sum_{j=1, j\neq i}^{m} (x_j - x_i)\frac{q_i \times q_j}{||x_j - x_i||^2}; & f(x_j) > f(x_i) \\ \displaystyle\sum_{j=1, j\neq i}^{m} (x_i - x_j)\frac{q_i \times q_j}{||x_j - x_i||^2}; & f(x_j) \leq f(x_i) \end{cases} \tag{15}$$

As we can see from (15), the resulting force vector $F_i = (F_1^i, F_2^i, \ldots, F_n^i)$ on the point x_i is the sum of force vectors induced by all other points that are neighbors to the x_i.

In the movement step of the EM algorithm, each point $x_i = (x_1^i, x_2^i, \ldots, x_n^i)$ except for the best one is moved in the direction of total force. The total force influencing each EM point is exerted by a step, the length of λ. In each generation, for each EM point, λ is randomly chosen from the interval [0, 1] and the same value of λ is used for each dimension of the current point. Formula (16) shows the movement equation of the vector x_i.

$$x_i^k = \begin{cases} x_i^k + \lambda\frac{F_i^k}{||F_i||}(1 - x_i^k), & F_i^k > 0 \\ x_i^k + \lambda\frac{F_i^k}{||F_i||}\cdot x_i^k, & F_i^k < 0 \end{cases} \tag{16}$$

x_i^k is the kth coordinate of ith particle and F_i^k is the calculated force on the kth coordinate of the ith particle.

5. Experimental results

In this section we present and discuss computational results of the proposed EM method. The EM implementation was coded in C programming language and compiled in Visual Studio 2010. All tests were carried out on the Intel Xeon E5410, @2.34 GHz. There are two groups of instances used for the testing of the EM algorithm.

The first group, named SAV, contains instances introduced in [14] and also used in [15]. The set of SAV instances contains a total of 22 problems. The instances include a different number of elements in set S (N = 10, 11, 12, 15, 20, 30, 50) and a different number of triples in C (ranging from 20 to 1000). For each N < 50, three instances are considered, and for N = 50 four instances are used. The name of each instance reflects the problem dimension: for example, the instance "11-100" indicates that set S has 11 elements and that collection C has 100 triples from the set S. The results obtained by the proposed EM are compared to the results obtained by the GAs (with and without local search), described in [15,9].

Our experiments are furthermore extended by using the real problem instances, named SLO, from [3,4]. In other to gather these instances, we used the RHMAPPER software package, a tool for creating genome maps developed at the Whitehead Institute/MIT Center for Genome Research. Inside the software distribution package, there is a set of markers from chromosome 18, as well as the complete set of mapped markers from the Whitehead's May 1996 release. We used this set of markers and the RHMAPPER command find_triples to generate triples of markers. At the end, we collected a total of 9 problem instances to solve with our algorithm. Seven of the 9 SLO instances can be considered as middle ones, containing from 15 to 25 elements with 120 to 478 triples. The remaining two instances are larger, containing 33 and 47 elements, with 1310 and 2888 triples, respectively. The name of each instance reflects the problem dimension in the same way as it does in the case of SAV instances. In mentioned papers [3,4], the focus is in determining the total ordering relation between markers, in order to find the path of markers of the maximal length. To solve that problem, the authors developed an algorithm for solving the variant of the MBP. After the algorithm is executed, markers that do not conform to total ordering relation are removed and the path of maximal length is found based on the satisfied betweenness constraints. Therefore, direct comparison between the proposed EM and the algorithm described in [4] is not possible.

Due to the fact that experiments from [14,15] are repeated 20 times, we decided to adopt the same scheme: for each instance, the algorithm is run 20 times, with different random seeds.

In order to precisely show the performances of our algorithm and also to make as fair comparison to other methods as possible, we made two classes of experiments. In the first class of the experiments, for both set of instances, we set the stopping criteria as follows: maximum of 100 iterations reached or 20 iterations without changing the best solution. For all instances except the largest one, 20 EM points are used, and 50 for the largest one.

The second class of the experiments is performed only on the SAV instances. Depending on the instances' size, we adjusted the stopping criteria and the number of EM points to match the fitness evaluation steps (operations) from [15]. The motivation behind this approach is in the fact that in cases where algorithms use local search procedures, equal conditions cannot be gained by only setting the equal number of generations [25]. Considering the additional fact that inside the local searches, both algorithms (EM and GA with LS from [15]) only partially (but in different ways) calculate the fitness, a simple counting of fitness evaluations would, again, not lead to a fair comparison. In order to overcome this specific problem we decided to count the total number of operations performed during the fitness calculations. This approach appears to be more general because it takes into consideration different implementations of the fitness functions. Naturally, the operations in both algorithms are counted in a consistent way. In order to determine the total number of operations performed during the fitness calculation of the GA with LS, we repeated all experiments from [15], counting that number. So, the stopping criteria for the second class of experiments are justified in a way that the total number of operations during the fitness calculation is approximately the same as for the GA with LS from [15].

Tables 1 and 2 provide the results of the first class of the experiments, obtained by the proposed EM on SAV and SLO instances, respectively. Both tables are organized in the same way. The first three columns contain the instance name, also indicating the problem's dimension, optimal solution, if it is known, and the best known solution from the literature in cases when optimal solution is not known. The best solution obtained by EM in 20 runs is given in the fourth column, named *EM*(*best*). New best solutions, achieved exactly by the proposed EM are marked as *new*. The average running time (t) used to reach the final EM solution for the first time is given in the fifth column, while the sixth and the seventh column (t_{tot} and *iterLS*) show the average total running time

Table 1
Experimental results of the proposed EM on SAV instances.

Inst.	Opt	*Best*	EM (best)	t (s)	t_{tot} (s)	*iterLS*	*SR* (%)	*ABSol*	agap (%)	σ (%)	
10–20	16	16	16	0.0017	0.03675	2335.4	100	16	0	0	
10–50	29	29	29	0.00505	0.06695	2511.3	100	29	0	0	
10–100	50	50	50	0.01515	0.13505	2901.3	100	50	0	0	
11–20	14	14	14	0.0014	0.0325	2138.4	100	14	0	0	
11–50	33	33	33	0.02135	0.0968	3765.4	100	33	0	0	
11–100	55	55	55	0.0137	0.16275	3505.4	100	55	0	0	
12–20	17	17	17	0.0056	0.0415	2843.6	100	17	0	0	
12–50	34	34	34	0.02605	0.106	4111.5	95	33.95	0.147	0.642	
12–100	56	56	56	0.0366	0.20255	4153.8	100	56	0	0	
15–30	26	26	26	0.01965	0.0805	4272	10	24.75	4.808	2.166	
15–70	–	46	46	0.04615	0.1978	5493.4	100	46	0	0	
15–200	–	106	106	0.21425	0.71025	7139.9	60	105.6	0.377	0.464	*New*
20–40	37	37	37	0.0478	0.16255	6424.5	10	35.45	4.189	2.087	
20–100	–	67	67	0.22035	0.5486	9759.6	35	66.3	1.045	0.84	*New*
20–200	–	116	116	0.38635	1.2053	10 872	35	115.05	0.819	0.751	*New*
30–60	55	55	54	0.14755	0.4432	11 125.9	5	52.01	3.519	1.474	
30–150	–	111	111	0.6347	1.58475	16 910	10	104.6	5.766	1.898	
30–300	–	185	185	1.3495	3.82755	19 566.7	5	178.1	3.73	1.064	*New*
50–100	–	87	87	0.65595	1.7709	20 784.7	15	85.45	1782	1.077	*New*
50–200	–	153	153	2.0437	4.9498	29 813.5	10	147.2	3.791	1.766	*New*
50–400	–	265	259	4.32465	12.205	34 996.8	10	252.25	2.606	1.033	
50–1000	–	536	536	67.0349	133.796	141 297.3	5	524	2.239	0.83	*New*

and the average number of local search steps for finishing the EM, respectively.

The solution quality in all 20 executions ($i = 1, 2, \ldots, 20$) is presented in the last four columns. The columns named *SR* and *ABSol* contain information about success rate and the best averaged solutions, obtained in 20 executions of the algorithm. Success rate is calculated by the formula $SR = \frac{NoB}{20} \cdot 100$ where *NoB* is the number of obtained optimal or best solutions in the total of 20 executions. *ABSol* is the averaged value of all obtained results. Last two columns of Tables 1 and 2, named *agap* and σ, provide the values of average gap and standard deviation. They are calculated in the following way: $agap = \frac{1}{20}\sum_{i=1}^{20} gap_i$, where $gap_i = 100 \cdot \frac{Opt.sol - sol_i}{Opt.sol}$ is evaluated with respect to the optimal solution $Opt.sol$, or the best-known solution $Best.sol$, i.e. $gap_i = 100 \cdot \frac{Best.sol - sol_i}{Best.sol}$ in cases where no optimal solution is found (sol_i represents the EM solution obtained in the i-th run). Standard deviation of the average gap $\sigma = \sqrt{\frac{1}{20}\sum_{i=1}^{20}(gap_i - agap)^2}$ is also presented.

Results shown in Table 1 clearly indicate that the proposed EM reaches all known optimal solutions, except one. For all other instances, the proposed EM algorithm achieves the same or better results than the current best known, except for two instances. Computational time for smaller instances (up to 20 elements and 200 constraints) is less than 1 s for all instances. For medium and large scale instances, computational time is less than 12 s, with the exception of the largest instance, for which the computational time is about 130 s. Longer execution time for the largest instances is expected, because a larger number of EM points are used (50 EM points vs. 20 points for all other instances). Short execution times indicate that the proposed EM is relatively fast. Even for very large problem instances the EM algorithm finds good quality solutions in reasonable times, indicating that it could be used for solving other large instances of similar dimensions.

The columns *SR*, *ABSol* and *agap* show that for all the problems with up to 15 elements and 200 constraints (except one), the EM reaches optimal solution in each of the 20 runs (in these cases, the success rate is 100%, averaged best solution is equal to optimal and the value *agap* is equal to 0). A small difference between averaged best solutions and best solutions, as well as rather small values of average gap and σ for medium and large scale instances indicate the reliability of the proposed algorithm.

From Table 2 it is evident that the EM easily finds optimal solutions (which were verified by CPLEX) for all of the seven middle scale instances. As it can be seen from the columns *SR* and *agap*, the algorithm achieves optimal solution in all 20 runs. For the two largest real instances, CPLEX could not find an optimal solution in less than 7200 s, so the optimality of the solutions obtained by the EM could not be verified. It is evident that, for these two instances, the EM achieves high quality solutions in a short period of time, which is less than 55 s for the largest instance.

Comparing the obtained results of both sets of instances, shown in Tables 1 and 2, it seems that the real instances are easier to solve than the artificial ones. This assumption can be based on the fact that triples contained in the real instances are more likely to preserve transitivity, which, in general, increases the objective function and simplifies the search. As a consequence, execution times of the proposed EM are shorter for the real instances than for the artificial ones of similar dimension.

Table 3 presents the first set of comparative results of CPLEX, best and averaged best results of the GA without local search given

Table 2
Experimental results of the proposed EM on SLO instances.

Inst.	Opt	*Best*	EM (best)	t (s)	t_{tot} (s)	*iterLS*	*SR* (%)	*ABSol*	agap (%)	σ (%)
15–120	118	118	118	0.0042	0.11485	7220.3	100	118	0	0
16–142	142	142	142	0.00585	0.1682	8537.3	100	142	0	0
19–187	176	176	176	0.00985	0.2644	9375.1	100	176	0	0
20–259	257	257	257	0.01765	0.4302	13 690.8	100	257	0	0
24–436	427	427	427	0.0492	1.2161	18 088.9	100	427	0	0
25–305	305	305	305	0.04765	0.85925	17 904.3	100	305	0	0
25–478	477	477	477	0.09525	1.40615	20 204.5	100	477	0	0
33–1310	–	1285	1285	0.6533	8.48835	40 408.1	100	1285	0	0
47–2888	–	2785	2785	7.06745	54.7091	77 093.6	100	2785	0	0

Table 3
Comparative results and running times of CPLEX, GA without LS, GA with LS and the proposed EM (the first class of experiments) on SAV instances.

Instance	*CPLEX*		*GA*			*GA + LS*			*EM*		
	sol	*t* (s)	best	*avg*	*t* (s)	best	avg	*t* (s)	best	avg	*t* (s)
10–20	16	0.437	16	15.75	0.194	16	15.8	0.088	16	16	0.037
10–50	29	7203.8	29	28.95	0.195	29	29	0.09	29	29	0.067
10–100	42	7201.6	50	48.79	0.652	50	48.75	0.116	50		0.135
11–20	14	2.125	14	13.65	0.2	14	13.65	0.089	14	14	0.032
11–50	33	7203.6	33	32.25	0.214	33	32.25	0.095	33	33	0.097
11–100	55	7201.7	55	53.55	0.243	55	53.55	0.115	55	55	0.163
12–20	17	1.156	17	16.6	0.197	17	16.7	0.09	17	17	0.042
12–50	32	7203.8	33	32	0.228	33	32	0.104	34	33.95	0.106
12–100	54	7202.2	56	54.25	0.246	56	54.35	0.119	56	56	0.203
15–30	26	3.172	25	22.75	0.217	25	22.9	0.101	26	24.75	0.08
15–70	45	7202.8	46	43.95	0.231	46	44.15	0.11	46	46	0.198
15–200	98	7201.4	105	102.85	0.289	105	102.85	0.149	106	105.6	0.71
20–40	37	1.625	36	32.3	0.34	37	32.8	0.171	37	35.45	0.163
20–100	63	7201.8	65	62.1	0.398	66	62.9	0.268	67	66.3	0.549
20–200	111	7201.1	113	111.6	0.4	114	111.9	0.225	116	115.05	1.205
30–60	55	7201.8	51	47.75	0.538	53	48.7	0.341	54	52.01	0.443
30–150	105	7200.8	102	95.65	0.627	111	98.5	0.598	111	104.6	1.585
30–300	165	7200.6	173	164.7	0.749	179	167.7	1.002	185	178.1	3.828
50–100	84	7200.9	84	78	1.147	86	81.25	1.163	87	85.45	1.771
50–200	154	7200.4	140	132.1	1.385	151	143.75	3.837	153	147.2	4.95
50–400	225	7200.3	240	230.15	1.535	265	248	7.8	259	252.25	12.2
50–1000	420	7200.2	504	482.9	2.169	532	514.15	19.86	536	524	133.8

in [14], the GA with local search [15] and, finally, best and averaged best results obtained in the first class of experiments using the proposed EM. For all approaches, running times are respectively provided. Upper time limit for CPLEX solver is set to 7200 s, which means that CPLEX is stopped after 2 h of execution, no matter if the current best found solution is not the optimal one.

From Table 3, it can be seen that in this class of the experiments, the proposed EM on medium and large instances outperforms all other approaches, comparing the best solutions, in six cases. For the instance 50–400 the GA with LS gives a better best result, while for the instance 50–200 CPLEX finds a better solution, which is not proved as optimal in 7200 s. Comparing the obtained averaged best solutions, we can see that EM outperforms other two methods for all instances. Computational times of the proposed EM approach are also comparable to other methods, especially with the GA with LS, which in [15] is proved to be a better method than the GA without LS. For large instances, the GA approach without LS is faster than the proposed EM, but EM provides better results for all these instances.

As mentioned before, the second class of experiments is performed under the equal number of operations used during the fitness evaluations, as in GA with LS from [15]. Since the number of fitness evaluations in GA without LS was less than in GA with LS (with partial fitness evaluations inside the LS counted), the comparison is made only with the latter algorithm. The obtained results and the appropriate data needed for the comparison are shown in Table 4, which is organized as follows: the first column is the instance name; the next five columns contain execution information for the GA with LS from [15]: averaged total execution time, averaged number of operations during the fitness evaluations, the best found and the averaged best solution, as well as the percentage gap. In the next seven columns, data related to the EM is shown. The first two columns represent the total number of EM points used, and the maximal allowed number of iterations with the unchanged objective value. These two columns show the way the EM algorithm was parameterized in order to achieve approximately the same number of operations as in [15]. The next five columns contain EM execution information organized in the same way as those for GA with LS. The last column shows the ratio between the operations counts inside EM and GA with LS. As we can see from Table 4, for each instance, this number is close to 1.

It can be seen from Table 4 that execution of the EM under approximately equal number of fitness operations did not decrease the quality of the achieved results. The execution times are less than in the first class of the experiments except for the last instance, where the execution time is now two times longer. The overall look on the obtained solutions indicates that they are not worse than those shown in Table 1. From the columns named *avg*, containing the averaged best solutions, we can see that the EM gives better results than the GA with LS for all instances. The execution times of these two methods are comparable for all instances except the last one.

In order to further investigate the statistical significance of results, a comprehensive statistical analysis has been made. Although the first class of the experiments cannot be a completely reliable base for a fair comparison, in order to include the GA without LS in statistical considerations, we firstly made a statistical analysis of the results obtained in the first class of the experiments. Fig. 5 shows a multiple-boxplot which enables a visual comparison of the performance of all three methods. Fig. 5 reinforces the idea that results are different and the proposed EM method is performing better than the rest. As expected, it is also clear from this figure that the GA without local search performs worse than the two other methods containing local search procedures.

The Shapiro–Wilk Test is performed, in order to investigate if the presented data has normal distribution. Since the answer is negative, the ANOVA test would not give reliable results. Thus, the non-parametric Kruskal–Wallis H Test is applied. The null hypothesis states that there is no significant difference between the three methods, with significance level $\alpha = 0.05$. Test results indicate that there is a statistically significant difference between the performances of algorithms ($H(2) = 14.928$, $P = 0.001$) with a mean rank of 20.61 for EM, 40.11 for GA + LS and 39.77 for GA.

For further analysis, the GA without local search is excluded and new statistical analysis is based on the data obtained under the equal conditions, as it is shown in Table 4. Again, the Kruskal–Wallis H Test is applied on two methods: GA with LS, from [15] and the proposed EM. For that test, the null hypothesis states that there is no significant difference between EM and GA with local search and significance level $\alpha = 0.05$.

Test results indicate that there is a statistically significant difference between the performances of algorithms ($H(1) = 8.142$,

Table 4
Comparative results of GA with LS and the proposed EM on SAV instances, under the equal number of operations inside the fitness evaluations.

Instance	GA+LS					EM							ratio
	t (s)	oper	best	avg	agap	NoP	maxRep	t (s)	oper	best	avg	agap	
10–20	0.088	560 484.6	16	15.8	1.25	10	9	0.0181	568 263.9	16	16	0	1.01
10–50	0.090	1 442 706.1	29	29	0	10	10	0.0244	1 583 080.5	29	29	0	1.1
10–100	0.116	3 212 922.9	50	48.75	2.5	10	8	0.0249	3 001 875.9	50	49.85	0.3	0.93
11–20	0.089	761 383.8	14	13.65	2.5	10	13	0.0068	709 882.5	14	14	0	0.93
11–50	0.095	1 619 781.2	33	32.25	2.273	10	7	0.0243	1 696 278.6	33	32.6	1.21	1.05
11–100	0.115	3 648 041.3	55	53.55	2.636	10	8	0.0438	3 696 641	55	54.95	0.09	1.01
12–20	0.090	1 052 076.7	17	16.7	1.765	10	12	0.0162	1 055 922.3	17	16.9	0.59	1.00
12–50	0.104	1 749 678.5	33	32	3.03	10	6	0.0180	1 580 584.3	34	33.15	2.5	0.90
12–100	0.119	3 426 367.6	56	54.354	2.946	10	5	0.0429	3 202 044.5	56	55.75	0.45	0.93
15–30	0.101	2 617 723.2	25	22.9	8.4	10	15	0.0328	2 641 266.9	26	24.4	6.15	1.01
15–70	0.110	4 153 530.8	46	44.15	4.022	10	8	0.0464	4 135 974.1	46	45.2	1.74	1.00
15–200	0.149	11 179 574.7	105	102.85	2.048	10	5	0.0996	10 813 652.6	106	104.5	1.41	0.97
20–40	0.171	3 783 754.6	37	32.8	11.351	10	10	0.0461	3 802 255.6	37	34.85	5.81	1.00
20–100	0.268	9 199 582.7	66	62.9	4.697	10	6	0.0836	9 831 588.9	66	64.9	1.67	1.07
20–200	0.225	32 551 303.9	114	111.9	1.842	15	9	0.2377	35 192 469.7	117	114.05	2.52	1.08
30–60	0.341	14 070 089.2	53	48.7	8.113	15	8	0.1019	14 116 349.3	53	51.5	2.83	1.00
30–150	0.598	68 902 867.6	111	98.5	11.261	15	10	0.4584	71 535 706.1	111	103.25	6.98	1.04
30–300	1.002	220 756 470.1	179	167.7	6.313	15	12	1.2411	212 252 945.1	185	177.25	4.19	0.96
50–100	1.163	140 689 692.7	86	81.25	5.523	20	17	0.8147	150 446 771.6	87	85.6	1.61	1.07
50–200	3.837	306 585 058.1	151	143.75	4.801	20	10	1.7494	302 706 700.4	153	146.2	4.44	0.99
50–400	7.800	1 061 632 848.5	265	248	6.415	20	22	7.8683	1 046 000 244	256	252.1	1.52	0.99
50–1000	19.854	2 786 825 663.5	532	514.15	3.355	50	75	267.6939	2 981 935 341	536	5254	1.98	1.07

$P = 0.004$) with a mean rank of 16.98 for EM and 28.02 for GA + LS. The graphical depiction of the results obtained by this test is shown in Fig. 6.

In order to compare the behavior of the improved local search to the existing local search, developed in [15], we extended the first class of experiments and developed the local search in the same way described in [15]. This local search was applied instead of our local search with caching, preserving the same control parameters as in the first class of the experiments. This part of the experiment was performed only on SAV instances, which are assumed to be more difficult. As we expected, the execution time of the EM with that variant of local search is increased by up to two times.

Comparison of the execution times of these two approaches for all SAV instances is provided in Table 5. The first column contains the names of instances, the second one, named t_{LS} [15], contains execution times of the EM with LS used in [15], while the third column, named t_{ImLS}, contains execution time of the EM with improved local search that uses caching, proposed in this paper. The next column contains the average number of LS calls. Last column indicates the ratio between the two corresponding values. Note that the only difference between these two approaches is in the caching technique, influencing only the execution time. So, all other obtained results, including the average number of LS calls are the same. From Table 5 we can see that the LS used in [15] is up to two times slower

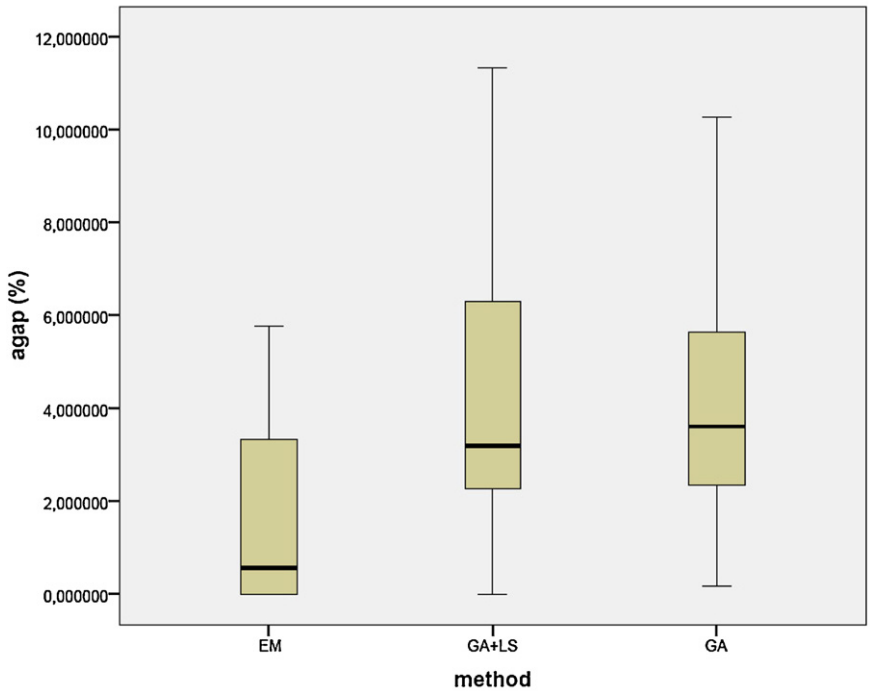

Fig. 5. Multiple box-plot for comparison all three methods.

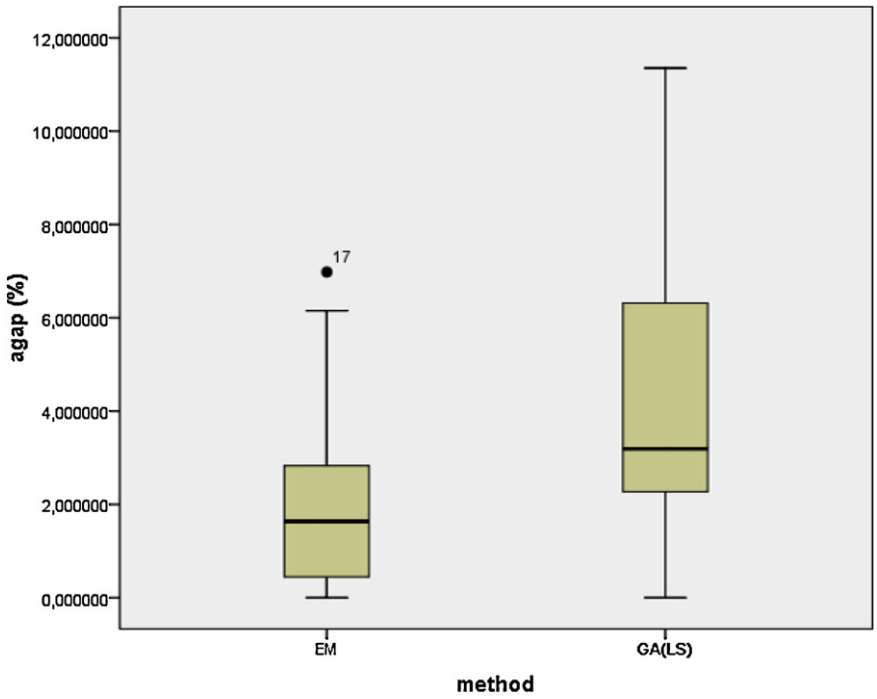

Fig. 6. Multiple box-plot for the comparison of two methods with local search.

Table 5
Execution time comparison of two LS approaches – SAV instances.

Instance	t_{LS} [15]	t_{ImLS}	Iter	Ratio
10–20	0.04515	0.03675	23 354	1.228571429
10–50	0.0946	0.06695	2511.3	1.412994772
10–100	0.195	0.13505	2901.3	1.443909663
11–20	0.0428	0.0325	2138.4	1.310769231
11–50	0.14145	0.0968	3765.4	1.461260331
11–100	0.23875	0.16275	3505.4	1.466973886
12–20	0.056	0.0415	2843.6	1.34939759
12–50	0.1632	0.106	4111.5	1.539622642
12–100	0.32625	0.20255	4153.8	1.610713404
15–30	0.1203	0.0805	4272	1.494409938
15–70	0.32255	0.1978	5493.4	1.630687563
15–200	1.15875	0.71025	7139.9	1.631467793
20–40	0.26705	0.16255	6424.5	1.642879114
20–100	0.9452	0.5486	9759.6	1.722931097
20–200	2.07085	1.2053	10 872	1.71811997
30–60	0.8014	0.4432	11 125.9	1.808212996
30–150	2.98515	1.58475	16 910	1.883672504
30–300	7.2224	3.82755	19 566.7	1.886951183
50–100	3.3291	1.7709	20 784.7	1.879891581
50–200	9.74945	4.9498	29 813.5	1.969665441
50–400	24.0207	12.205	34 996.8	1.968103236
50–1000	258.02715	133.796	141 297.3	1.928511689

than the improved LS with caching. Also, it can be concluded that the ratio between these two execution times increases as the problem dimension increases and approaches to 2. This fact verifies the consideration presented in the previous section, where we noted that the LS from [15] deals with a list of satisfied constraints which is updated twice in each iteration, while our approach, improved LS with caching, uses the structure holding only the information of the total number of satisfied constraints, and that structure is updated only once in each iteration.

6. Conclusions

In this paper, we describe an EM metaheuristic for solving the Maximum Betweenness Problem. A new encoding scheme is used, which gives a suitable representation of an individual EM point. The specific encoding scheme enables fast and efficient transformation from the continuous space of EM points to the discrete space of permutations and vice versa, following the idea that minor movements of EM points should not change the objective value. The algorithm uses an effective 1-swap based improved local search procedure, which implements the caching technique.

Computational experiments are performed on real and artificial instances from the literature. In order to show best performances of the proposed EM, but also to meet equal conditions for fair comparison, two classes of experiments are performed. The results achieved by the first class of the experiments show that the proposed EM achieves all known optimal solutions with the exception of one instance. For all medium and large scale instances, except two, the proposed EM algorithm gives better results than the current best ones. The computational times for executing the algorithm are

comparable to executing times of other approaches. A rather small average gap and a standard deviation confirm the reliability of the proposed method. The second class of the experiments indicates that the proposed EM outperforms other approaches, which is also confirmed by the statistical analysis.

Additional tests are made to examine the behavior of the proposed local search procedure. Experiments show that our improved local search which uses a cached structure for storing information about a number of satisfied betweenesses of each element is up to two times faster than the existing local search used in previous approaches.

The EM implementation described in this paper can be extended in several ways. Researches could concentrate on the acceleration of the algorithm by taking advantage of parallel computation, and on the EM hybridization with exact methods or other heuristics.

Acknowledgement

This research is partially supported by the Ministry of Science, Technology and Development, Republic of Serbia, under the Project 174010: Mathematical Models and Optimization Methods for Large-Scale Systems.

References

[1] J. Opatrny, Total ordering problem, SIAM Journal on Computing 8 (1979) 111–114.
[2] B. Chor, M. Sudan, A geometric approach to betweenness, SIAM Journal on Discrete Mathematics 11 (4) (1998) 511–523.
[3] D. Slonim, D. Stein, L. Kruglyak, D. Lander, Rhmapper: an interactive computer package for constructing radiation hybrids maps, 1996. URL http://www.genome.wi.mit.edu/ftp/pub/software/rhmapper (last accessed 20.11.11).
[4] D. Slonim, D. Stein, L. Kruglyak, E. Lander, Building human genome maps with radiation hybrids, Journal of Computational Biology 4 (1997) 487–504.
[5] D. Cox, M. Burmeister, E. Price, S. Kim, R. Myers, Radiation hybrid mapping: a somatic cell genetic method for constructing high resolution maps of mammalian chromosomes, Science 250 (1990) 245–250.
[6] T. Christof, M. Junger, J. Kececioglu, P. Mutzel, G. Reinelt, A branch-and-cut approach to physical mapping with end-probes, in: Proceedings of the 1st Annual International Conference on Computational Molecular Biology (RECOMB-97), 1997, pp. 84–92.
[7] G. Goss, H. Harris, New methods for mapping genes in human chromosomes, Nature 255 (1975) 680–684.
[8] T. Christof, M. Oswald, G. Reinelt, Consecutive ones and a betweenness problem in computational biology, in: Proceedings of the 6th Conference on Integer Programming and Combinatorial Optimization, vol. 1412, Springer-Verlag Lecture Notes in Computer Science, 1998.
[9] A. Savić, J. Kratica, M. Milanović, D. Dugošija, A mixed integer linear programming formulation of the maximum betweenness problem, European Journal of Operational Research 206 (3) (2010) 522–527.
[10] C.H. Papadimitriou, M. Yannakakis, Optimization, approximation and complexity classes, Journal of Computer and System Sciences 43 (1991) 425–440.
[11] M.X. Goemans, F. Rendl, Semidefinite programming in combinatorial optimization, Mathematical Programming 79 (1997) 143–161.
[12] W. Guttmann, M. Maucher, Variations on an ordering theme with constraints, in: Proceedings Fourth IFIP International Conference on Theoretical Computer Science TCS, Springer, 2006, pp. 77–90.
[13] A. Goerdt, On random betweenness constraints, Lecture Notes in Computer Science 5699 (2009) 157–168.
[14] A. Savić, On solving of maximum betweenness problem using genetic algorithms, Serdica Journal on Computing 3 (3) (2009) 299–308.
[15] A. Savić, J. Kratica, J. Fijuljanin, Hybrid genetic algorithm for solving of maximum betweenness problem, in: Proceedings of the 1st International and 10th Balcan Conference on Operational Research – BALCOR 2011, 2011, pp. 402–408.
[16] CPLEX solver, IBM company. URL http://www.ibm.com/software/integration/optimization/cplex-optimizer
[17] C. Blum, J. Puchinger, G.R. Raidl, A. Roli, Hybrid metaheuristics in combinatorial optimization: a survey, Applied Soft Computing 11 (6) (2011) 4135–4151.
[18] C. Blum, A. Roli, Metaheuristics in combinatorial optimization: overview and conceptual comparison, ACM Computing Surveys 35 (3) (2003) 268–308.
[19] S.I. Birbil, S.C. Fang, An electromagnetism-like mechanism for global optimization, Journal of Global Optimization 25 (2003) 263–282.
[20] S.I. Birbil, S.C. Fang, R.L. Sheu, On the convergence of a population-based global optimization algorithm, Journal of Global Optimization 30 (2004) 301–318.
[21] Z. Naji-Azimi, P. Toth, L. Galli, An electromagnetism metaheuristic for the unicost set covering problem, European Journal of Operational Research 205 (2) (2010) 290–300.
[22] V. Filipović, An electromagnetism metaheuristic for the uncapacitated multiple allocation hub location problem, Serdica Journal on Computing 5 (3) (2011) 261–272.
[23] E. Cuevas, D. Oliva, D. Zaldivar, M. Pérez-Cisneros, H. Sossa, Circle detection using electro-magnetism optimization, Information Sciences 182 (1) (2012) 40–55.
[24] C. Su, H. Lin, Applying electromagnetism-like mechanism for feature selection, Information Sciences 181 (5) (2011) 972–986.
[25] C. Črepinšek, S. Liu, L. Mernik, A note on teaching-learning-based optimization algorithm, Information Sciences 212 (2012) 79–93.