Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 346 (2019) 449–460

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

A Multi-agent Transgenetic Algorithm for the Bi-objective Spanning Tree Problem

Islame F. C. Fernandes,[1,2] Silvia M. D. M. Maia,[3]
Elizabeth F. G. Goldbarg,[4] Marco C. Goldbarg[5]

Department of Informatics and Applied Mathematics
Federal University of Rio Grande do Norte (UFRN)
Natal, Brazil

Abstract

The Bi-objective Spanning Tree (*BiST*) is an NP-hard extension of the Minimum Spanning Tree (*MST*) problem. The *BiST* models situations in which two conflicting objectives need to be optimized simultaneously. The *BiST* has been studied in the literature and several heuristic algorithms were proposed for it, most of them evolutionary techniques. The transgenetic algorithms are among these evolutionary techniques which were successfully applied to the *BiST*. However, *a priori* defined parameters can limit the search mechanisms used within the algorithm. In this study, we propose a new transgenetic algorithm for the *BiST* in which the decision about the search mechanisms used along its execution is automatically made. An analysis of the results of computational experiments carried on 165 benchmark instances showed that the algorithm proposed in this study produces good approximation sets concerning two different quality indicators.

Keywords: Bi-objective Spanning Tree, Evolutionary Algorithms, Combinatorial Optimization

1 Introduction

The multi-objective spanning tree with sum objectives (*MoST*) is an NP-hard extension of the Minimum Spanning Tree (*MST*) problem. Let $G = (V, E)$ be an undirected connected graph, where $V = \{v_1, v_2, ..., v_n\}$ denotes a finite set of vertices and $E = \{(i, j) \mid v_i, v_j \in V\}$ a finite set of edges such that $|E| = m$. Let $w : E \rightarrow \Re^q$ be a function that assigns a vector $w_{ij} = (w^1_{ij}, w^2_{ij}, ..., w^q_{ij})$, $q > 1$,

[1] This research was partially supported by the NPAD/UFRN (High Performance Computing Center at Universidade Federal do Rio Grande do Norte), CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) and by the CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico), Brazil, under grants 301845/2013-1 and 308062/2014-0.

[2] Email: islamifelipe@gmail.com

[3] Email: silvia@dimap.ufrn.br

[4] Email: beth@dimap.ufrn.br

[5] Email: marcocgold@gmail.com

https://doi.org/10.1016/j.entcs.2019.08.040

of non-negative real weights to $(i, j) \in E$. A spanning tree $T = (V, E_T)$ of G is an acyclic spanning subgraph of G, where $E_T \subseteq E$. Let X be the set of all spanning trees of G and $Z \subseteq \Re^q$ the objective space. Function $f : X \to Z$ associates each $T \in X$ with a vector $f(T) = (f_1(T), f_2(T), ..., f_q(T)) \in Z$, where $f_k(T)$, $k \in \{1, ..., q\}$, is defined by equation (1).

$$f_k(T) = \sum_{(i,j)\in E_T} w_{ij}^k \tag{1}$$

Let T_1 and $T_2 \in X$. T_1 weakly dominates T_2, denoted by $T_1 \preceq T_2$, if and only if $f_k(T_1) \leq f_k(T_2)$, $\forall k \in \{1, ..., q\}$. T_1 dominates T_2, denoted by $T_1 \prec T_2$, if and only if $T_1 \preceq T_2$ and $\exists j \in \{1, ..., q\}$ such that $f_j(T_1) < f_j(T_2)$. The set $X^* = \{T^* \in X \mid \nexists T \in X, T \prec T^*\}$ of non-dominated solutions is called Pareto optimal set. Set $Z^* = \{f(T^*) \mid T^* \in X^*\}$ is called Pareto optimal front. $T^* \in X^*$ is said to be an efficient solution and $f(T^*)$ an efficient objective vector or point. When solving the *MoST*, one may be interested in finding the Pareto optimal set, X^*, or the Pareto optimal front, Z^*. When $q = 2$, the *MoST* is called bi-objective spanning tree (*BiST*). This problem is NP-hard [1]. Efficient solutions can be classified into two categories: supported and non-supported [10]. An efficient solution is supported if and only if there exists a non-negative real-weight vector $\lambda = (\lambda_1, \lambda_2, ..., \lambda_q) \in \Re^q$, satisfying equation (2), from which that supported solution can be computed by solving equation (3). An efficient solution which cannot be computed by solving equation (3) for any weight vector $\lambda \in \Re^q$ is called non-supported. The *BiST* supported solutions can be computed by a geometric method, called dichotomous search, proposed by [13]. That method executes in polynomial time if we want to compute only the set of supported points [29], and in this case, the major challenge is to compute those points related to non-supported solutions.

$$\sum_{k=1}^{q} \lambda_k = 1 \tag{2}$$

$$\min_{T\in X} \sum_{k=1}^{q} \lambda_k f_k(T) \tag{3}$$

Exact and heuristic methods were proposed for the *BiST* and the *MoST*, most of them are described in the survey presented by [27]. Exact algorithms include a generalization of the Prim's algorithm for the single-objective case [7], branch-and-bound [24] [29], k-best [30] and dynamic programming [22]. Among the metaheuristics, evolutionary algorithms have been widely applied to multi-objective problems. Some evolutionary algorithms proposed for the *MoST* include: genetic algorithms [31] [5] [19], memetic algorithms [25], evolutionary strategies [9], and transgenetic algorithms [20].

The transgenetic algorithm proposed by [20], called *TLP*, is an effective approach for the *BiST*. The *TLP* was compared to a state-of-art algorithm proposed by [2] and produced high quality results. Transgenetic algorithms are evolutionary approaches that simulate natural mechanisms of horizontal gene transfer and endosymbiosis [12]. The metaphor is based on the exchange of genetic material be-

tween a host cell and a population of endosymbionts by the action of agents which are called transgenetic agents. Basically, a transgenetic algorithm maintains a population of solutions (endosymbionts), a repository of information to be used along the search (host cell), and transgenetic agents. The latter manipulates the endosymbionts, modifying the solutions. There are different classes of transgenetic agents and some of them use information from the repository. The *TLP* uses two types of transgenetic agents: plasmids and transposons. They are described in section 2. User-defined parameters control the decision of which of those agents manipulate which individual of the population. Thus, for growing numbers of transgenetic agents, the number of parameters increases. It can be a burden for the algorithm. Besides, depending on the instances to which the algorithm is applied, these parameters may vary significantly from an instance to another. Since the work presented by [4], reactive mechanisms have been incorporated to metaheuristics to tune parameters automatically. In the transgenetic agents case, the algorithm learns which of them to apply at some point of the search. In this work, we propose a new transgenetic algorithm that deals with 10 transgenetic agents from the plasmid and transposon classes. The probability of choosing a transgenetic agent is related to its success rate, which may change along the search. The algorithm is called *TMA*.

We investigated the potential of the *TMA* to produce high-quality approximation sets in a computational experiment carried on 165 benchmark instances from three classes. The *TMA* is compared to the *TLP* and to the *NSGA-II* algorithm. The latter, proposed by [8], is a successful algorithm for bi-objective problems [28]. Computational experiments showed that the *TMA* outperforms both *TLP* and *NSGA-II* regarding solution quality and processing time. We also investigated the potential of each transgenetic agent tested in this study.

This work contains other four sections. The *TLP* and the *TMA* are described in sections 2 and 3, respectively. The results of the computational experiments are reported in section 4 and final conclusions are addressed in section 5.

2 *TLP*

In this section, we present the *TLP*, a transgenetic algorithm that inspired the one proposed in this study. The *TLP*, proposed by [20], is a transgenetic algorithm for the *BiST*. It maintains a population of endosymbionts and a limited global archive of non-dominated solutions. The global archive plays the role of the host's repository. Each solution of the archive is a source of genetic information. An endosymbiont is a spanning tree. The population has $\#popSize$ elements, 90% of them generated by a multi-objective greedy randomized version of the Prim's algorithm [21] (*rmcPrim*), adapted from [15]. The *RandomWalk* method [23] generated the remaining of the population. The *rmcPrim* method uses a scalarizing vector for building a solution. Let $c(e)$ be the scalarized cost of edge e, the edge that would be chosen by the Prim's algorithm in a given iteration. A restricted candidate list is built with the edges whose scalarized costs are, at most, $(1 + \beta)c(e)$, where $\beta \in [0, 1]$ is a user-defined parameter. Every iteration of the *rmcPrim* algorithm, an edge

is selected from the restricted candidate list, with uniform probability, to build a tree. During the generation of endosymbionts with the *rmcPrim* method, repeated endosymbionts are not accepted and dominated spanning trees are accepted with probability 60%. These restrictions do not apply when the population is generated with the *RandomWalk*. The latter builds a tree iteratively selecting, with uniform probability a random edge that connects a node already included in the tree with a node not yet included.

The *TLP* uses two types of transgenetic agents: plasmids and transposons. There are two types of plasmids: simple and recombinant. Plasmids consist of a piece of genetic information and a manipulation method. In the case of the *BiST*, the piece of genetic information is a set with pl edges, where pl is randomly chosen in range $[0.25n, 0.50n]$. The difference between the plasmid types lies on the way their genetic information is obtained. The genetic information of simple plasmids comes from a solution (source) of the external archive. The solution is chosen at random in the least crowded region of the objective space defined by the points correspondent to the solutions of the archive. In the *TLP*, the genetic information of a recombinant plasmid comes from a partial solution built by the *rmcPrim*. The methods used to build the information on the simple and recombinant plasmids are called, $s1$ and $s2$, respectively. Let p be an endosymbiont subject to the manipulation of a transgenetic agent. The manipulation method of the plasmids, called $m1$, builds a tree by first inserting the edges of the plasmid in an empty tree and then random edges of p, as much as possible, that do not induce a cycle. If necessary, random edges from the graph are added until obtaining a spanning tree. The endosymbiont resulting from the manipulation, $\overline{p}$, replaces p in the current population if $\overline{p} \prec p$ or if $\nexists p'$ in the global archive such that $p' \prec \overline{p}$.

Transposons consist of perturbing procedures that are applied to restricted areas of an endosymbiont, rather than to the entire solution. The *TLP* has two transposons called *remTransp* and *primTransp*. The disturbance method of *remTransp* consists in removing $\#sizeRem$ edges randomly chosen from p and adding new edges to p, where $\#sizeRem$ is a parameter. Each edge removed from p is replaced by an edge that reconnects the tree and whose scalarized cost is best among all possible edges. Each edge removal/addition operation generates a new tree. The new trees are stored in a local archive of non-dominated solutions. Finally, the *remTransp* returns a solution $\overline{p}$ randomly chosen from the local archive. The *primTransp* removes l edges from p, where l is an integer randomly chosen from the range $[\lfloor 0.9n \rfloor, \lfloor 0.95n \rfloor]$. The higher the scalarized cost of an edge the higher is its probability to be removed. Then, the *primTransp* rebuilds the solution with the *rmcPrim* method.

Every iteration of the *TLP*, all endosymbionts are submitted to a plasmid or a transposon. The plasmid type is chosen with probability $\#probPlasm$ (consequently, the probability of choosing a transposon is $1 - \#probPlasm$). The $\#probPlasm$ probability remains fixed for $\#intGerSet$ iterations of the *TLP* algorithm, then $\#probPlasm$ is increased. If the plasmid type is chosen, the algorithm chooses between the simple or recombinant type at ran-

dom with uniform probability. If the transposon type is chosen, the *remTransp* has probability $\#probRemTransp$ to be selected (*primTransp* has probability $1 - \#probRemTransp$), where $\#probRemTransp$ is a parameter.

The *TLP* was applied to benchmark instances proposed by [15] up to 1000 vertices. It outperformed previous algorithms for the *BiST* proposed by [26] and [2].

3 The Multi-agent Transgenetic Algorithm

The transgenetic agents are the tools to modify the solutions in the search performed by a transgenetic algorithm. Many transgenetic agents may exist, some of them directed to diversify and others to intensify the search. This section presents a transgenetic algorithm with automatic control of the decision of which agent is selected to manipulate an endosymbiont from the current population. Procedure *TMA* shows the general pseudo-code of the algorithm proposed in this study.

The *TMA* is derived from the *TLP* and maintains some of the latter's elements. The *TMA* maintains a limited archive of non-dominated solutions, G_A. To initialize G_A, a set with $\#numSup$ well distributed supported solutions is generated with the geometric method proposed by [13] (line 2), where $\#numSup$ is a parameter. Each supported solution is computed with a weight vector calculated by the geometric method. Each weight vector induces an ordering of the edges. The algorithm maintains the $\#numSup$ weight vectors and the correspondent lists of sorted edges for future computations. Thus, we consider that the information maintained in the host's repository consists of G_A, the list of supported solutions, the weight vectors and the lists of sorted edges. The reason for storing the list of supported solutions is that the G_A archive is limited and some solutions may be lost when it is updated.

The method used to create the initial population of the *TMA* is the one used by the *TLP*, described in section 2. There are 7 plasmids and 3 transposons implemented in the *TMA*. Their success rates are set to 0, initially (line 4).

Every iteration, a set with $\#numPlas$ plasmids is created (line 7, procedure *createPlasmids*). There are two types of plasmids: simple and recombinant. The algorithm deals with three simple plasmids and four recombinants. As in the *TLP*, the information of a plasmid is a set with *pl* edges. The information of simple plasmids is created by one from two methods: $s1$ described in section 2, and *s4*. Method *s4* selects *pl* random edges from a random supported solution. The information of recombinant plasmids was created by one from two methods: $s2$ described in section 2, and *s3*. The $s3$ method builds 40% of the plasmid's information by the $s1$ method and 60% by the $s2$. Two manipulation methods were implemented for the plasmids: $m1$ described in section 2, and $m2$. In the $m2$ method, the edges from p and the plasmid's information are joined and sorted regarding a weight vector chosen at random from the host's repository. A new solution, $\overline{p}$, is built by the Kruskal's method [17]. If $\overline{p}$ is not a tree, new edges from the original graph are added to $\overline{p}$ regarding the same weight vector and the Prim's method. Table 1 summarizes the methods used in the plasmids. The acceptance criterion implemented in the *TLP*

was also used in the *TMA* and implemented in procedure *better*().

Algorithm 1 *The TMA algorithm*

```
1:  procedure TMA(level : real; #numSup, #popSize, #numPlas, #GerReset
    : integer )
2:      preProcessingPhase(#numSup, G_A)
3:      generate_population(P = {p_1, ..., p_#popSize}, G_A)
4:      success[t] = 0, ∀ transgenetic agent t
5:      gen ← 1
6:      repeat
7:          setProbPlas();  setProbTrans();  createPlasmids(#numPlas)
8:          for all p ∈ P do
9:              if random(0,1)< level then
10:                 pl ←getPlasm(#numPlas);  p̄ ←plasmid(pl, p)
11:                 if better(p̄, p) then
12:                     p ← p̄;  success[pl]++
13:                 end if
14:             else
15:                 tr ←getTransp();  p̄ ← transpid(tr, p)
16:                 if better(p̄, p) then
17:                     p ← p̄;  success[tr]++
18:                 end if
19:             end if
20:         end for
21:         updateLevel(level)
22:         if gen mod #GerReset == 0 then
23:             success[t] = 0, ∀ transgenetic agent t
24:         end if
25:         gen + +
26:     until a stopping criterion is satisfied
27:     return G_A
28: end procedure
```

Table 1
Plasmids

Name	Type	Information	Manipulation	Name	Type	Information	Manipulation
$plm1$	Recombinant	$s2$	$m1$	$plm5$	Recombinant	$s3$	$m1$
$plm2$	Recombinant	$s2$	$m2$	$plm6$	Recombinant	$s3$	$m2$
$plm3$	Simple	$s1$	$m1$	$plm7$	Simple	$s4$	$m2$
$plm4$	Simple	$s1$	$m2$				

Three transposons were implemented: *newRemTransp*, *krusTransp* and *swapTransp*. The *newRemTransp* derives from the *remTransp*, presented in section 2. The difference between them is the method to select a solution to be returned. Both transposons create a list of non-dominated solutions. The *newRemTransp* may return the solution closest to the ideal point, concerning the Euclidean distance in the objective space or a random solution chosen with uniform probability. The probability that the former is returned is 70%. The *krusTransp* works likewise

primTransp, except that the Kruskal's method rebuilds the tree in the former. Finally, the *swapTransp* removes a random edge, e, from endosymbiont p, resulting in two connected components. Then, all trees induced by the inclusion of each edge in the cut-set between the two connected components are tested. This operation produces a set of pairwise non-dominated trees. The *swapTranp* returns a tree from this set which is chosen by the same method of the *newRemTransp*.

The G_A is updated whenever a solution $\overline{p}$ is created by a transgenetic agent and $\nexists p' \in G\_A$ such that $p' \prec \overline{p}$.

Every iteration, all endosymbionts are submitted to the attack of a transgenetic agent (lines 8-20) which is randomly selected. The probability that a plasmid is chosen is given by *level* (line 9) and, consequently, that a transposon is chosen is $1 - level$ (line 14). Initially, the probability of choosing a plasmid or a transposon is the same, i.e., $level = 0.5$. Procedure *updateLevel* (line 21) updates the value of variable *level* as follows. The value remains the same for 5.10^5 evaluations of the objective function. Then, in the remaining iterations, it is updated to $countEvel/10^6$, where *countEvel* is the current number of evaluations. As a consequence, the probability that a transposon is chosen decreases over time. This strategy was adopted since, as observed by [20], transposons seem to be more effective at initial iterations whereas plasmids are more effective in final iterations. Once the transgenetic agent type is chosen, plasmid or transposon, the algorithm selects from that class (lines 10 and 15, for a plasmid or transposon, respectively). The selection depends on the transgenetic agent's success rate in the last $\#GerReset$ iterations, where $\#GerReset$ is a user-defined parameter. A counter is maintained for each transgenetic agent. The counter is incremented whenever the manipulation of the transgenetic agent is successful, i.e., when procedure *better* returns a true value. Transgenetic agents whose counters are higher are more likely to be selected. A roulette wheel selection is used to choose the transgenetic agents. Initially, the same type agents are equally likely. Every iteration, the probability associated with the plasmids and transposons are updated, concerning their success rates, in line 7. After each sequence of $\#GerReset$ iterations, the success counters are set to zero, and the probabilities are reinitialized (line 23). Thereby, the *TMA* can deal with long-term (high $\#GerReset$) or short-term (low $\#GerReset$) success rate history.

4 Computational Experiments

The experiments were executed on the infrastructure provided by the High-performance Computing Center at UFRN (NPAD/UFRN). Each test was allocated to a core of an Intel Xeon processor E5-2698v3 with 2.3 GHz and 4Gb of RAM per core, running CentOS 6.5, 64 bits. The experiments were carried on 165 complete graphs generated by the method proposed by [15]. This set of instances is called *KNW*. The *KNW* set is divided into three classes: correlated (*Corr*), anti-correlated (*Anticorr*) and concave (*Conc*) instances. Each class contains 55 instances identified by $n.ID$, where n is the number of vertices, ranging from 50 to 1000, and $ID \in \{1, 2, 3, 4, 5\}$ which identifies instances generated with the same

parameters. Correlated instances identified with $ID = 1, 2, 3, 4, 5$ were generated with correlation factors $0.1, 0.3, 0.5, 0.7, 0.9$, respectively. Anti-correlated instances were similarly generated with the corresponding negative correlation factors. The parameters η and ζ, required to generate concave instances, were randomly chosen, respectively, from $[0.0009, 0.3]$ and $[0.0001, 0.03]$.

We compared the *TMA* to the *TLP* and to a classic successful genetic algorithm for bi-objective problems named Non-Dominated Sorting Genetic Algorithm (*NSGA-II*) proposed by [8]. The initial population P of the *NSGA-II* was created by the same method described in section 2. Solutions were encoded by the list of edges data structure. Every iteration, an offspring population, $\overline{P}$, such that $|\overline{P}| = |P|$, is created. A variation of the binary tournament [25] is used as the selection scheme. Two pairs of individuals are randomly selected in P: the first pair competes in the first objective and the second one in the second objective. One offspring is generated by the recombination and mutation operators suggested by [23]. The recombination operator consists in applying the *randomWalk* method to the union of the parental edge sets. The mutation adds a new edge to the tree and removes another from the cycle created. The individuals for the next generation are chosen by non-dominance rank and crowding distance as in the standard *NSGA-II*.

Thirty independent executions of each heuristic algorithm were performed for each instance. The stopping criterion was 10^6 evaluations for all algorithms. We analyzed processing times and the quality of the approximation sets produced. Two indicators were used to assess the quality of the approximation sets: hypervolume and inverted generational distance. The hypervolume (HV) indicator was proposed by [32] and is the only single set quality measure that is known to be strictly monotonic about Pareto dominance [3]. The HV measures the volume of the objective space dominated by an approximation set. To compute the HV, we need a reference point to bound the objective space. We used the method proposed by [16] to compute the reference point. The inverted generational distance (IGD) indicator [6] is used to assess the diversity of solutions in the approximation set. The IGD is computed with equation (4), where $Z^{*\prime}$ is an approximation set, R is a reference set, and $ds(r, Z^{*\prime})$ is the distance between a point $r \in R$ and the nearest objective vector in $Z^{*\prime}$. In this study, the reference set R of an instance was obtained by filtering the non-dominated points from the union of all approximation sets generated by all algorithms tested [16].

$$IGD(Z^{*\prime}, R) = \frac{\sum_{r \in R} ds(r, Z^{*\prime})}{|R|} \tag{4}$$

The Friedman test [11] was applied to verify significant differences among the solutions produced by the algorithms tested. The Holm posthoc test [14] was used to detect specific differences between the *TMA* and the other algorithms. The significance level was 0.05.

We used the values defined by [20] for the *TLP* parameters, including $\#sizeRem = 0.05n$ and $\beta = 0.03$ for the *TMA*. The other *TMA* parameters were tuned with the IRACE package [18]: $\#popSize = 150$, $\#numSup = 59$, $\#numPlas = 21$, and $\#GerReset = 196$. Since we wanted to give the same proba-

bility for the algorithm to choose plasmids or transposons in the first iteration, we set the *level* parameter to 0.5. The external archive used in the *TMA* was limited to 300 solutions and was updated with the adaptive grid technique as proposed by [15]. The crossover and mutation rates of the NSGA-II algorithm were tuned with the IRACE and are, respectively, 0.97 and 0.04. The population size adopted for the NSGA-II was 150, as the *TLP* and the *TMA*.

Table 2 shows the average rankings computed by the Friedman test for each group and class of *KNW* instances. The lower the value, the better the algorithm ranking. The average processing times, in seconds, are shown in column $T(s)$. All p-values computed by the Friedman test were less than 0.05, meaning that, statistically, there is significant difference among *TMA*, *TLP* and *NSGA-II* regarding the *HV* and the *IGD*. Except from the *HV* indicator computed for the anti-correlated $ID = 1$ group (red cells), the *TMA* algorithm ranked better than the other algorithms concerning the *HV* and the *IGD* (green cells).

For most cases where the *TMA* ranked better, the Holm posthoc test indicated significant differences. The exceptions for the *HV* indicator were: groups 1 and 5 of correlated instances, group 2 of anti-correlated instances and groups 1, 2, and 3 of concave instances. These exceptions concern the *TLP*. The exceptions for the *IGD* indicator concerning the *TLP* were: group 5 of correlated instances and groups 1, 2, and 3 of concave instances. Only one exception was observed for the *IGD* indicator concerning the *NSGA-II* algorithm, group 1 of anti-correlated instances and groups 1, 2, and 3 of concave instances.

Column $T(s)$ shows that, on average, the *TMA* spends less processing time than the other algorithms. We can conclude that the automatic decision about which transgenetic agent, based on success rates, improved the algorithm both regarding solution quality and processing time.

Table 2
Results for the KNW instances.

ID	Alg.	Corr.			Anti-corr.			Conc.		
		HV	IGD	T(s)	HV	IGD	T(s)	HV	IGD	T(s)
1	TMA	1.27	1.00	81.99	1.82	1.45	41.44	1.36	1.27	51.63
	TLP	1.73	2.00	194.72	1.27	2.82	198.42	1.91	2.00	172.32
	NSGA-II	3.00	3.00	513.06	2.91	1.73	579.21	2.73	2.73	547.15
2	TMA	1.00	1.00	91.77	1.09	1.18	36.88	1.27	1.27	58.35
	TLP	2.00	2.00	198.23	1.91	2.09	204.04	2.00	2.00	168.20
	NSGA-II	3.00	3.00	499.36	3.00	2.73	580.15	2.73	2.73	551.37
3	TMA	1.00	1.00	92.30	1.00	1.09	37.75	1.18	1.18	64.70
	TLP	2.00	2.00	195.22	2.00	2.09	203.65	2.00	2.00	176.33
	NSGA-II	3.00	3.00	486.20	3.00	2.82	582.27	2.82	2.82	546.83
4	TMA	1.00	1.00	92.76	1.00	1.00	43.79	1.00	1.00	67.63
	TLP	2.00	2.00	195.44	2.00	2.09	209.36	2.00	2.00	197.55
	NSGA-II	3.00	3.00	474.42	3.00	2.91	584.22	3.00	3.00	537.84
5	TMA	1.14	1.14	82.01	1.00	1.00	60.04	1.09	1.09	60.21
	TLP	1.95	1.95	188.60	2.00	2.00	209.06	2.00	2.00	186.78
	NSGA-II	2.91	2.91	458.88	3.00	3.00	582.73	2.91	2.91	633.06

The graphics of Figure 1 show the average success rate of plasmids (a-c) and transposons (c-d) for each *KNW* class. The success rate of an agent is computed

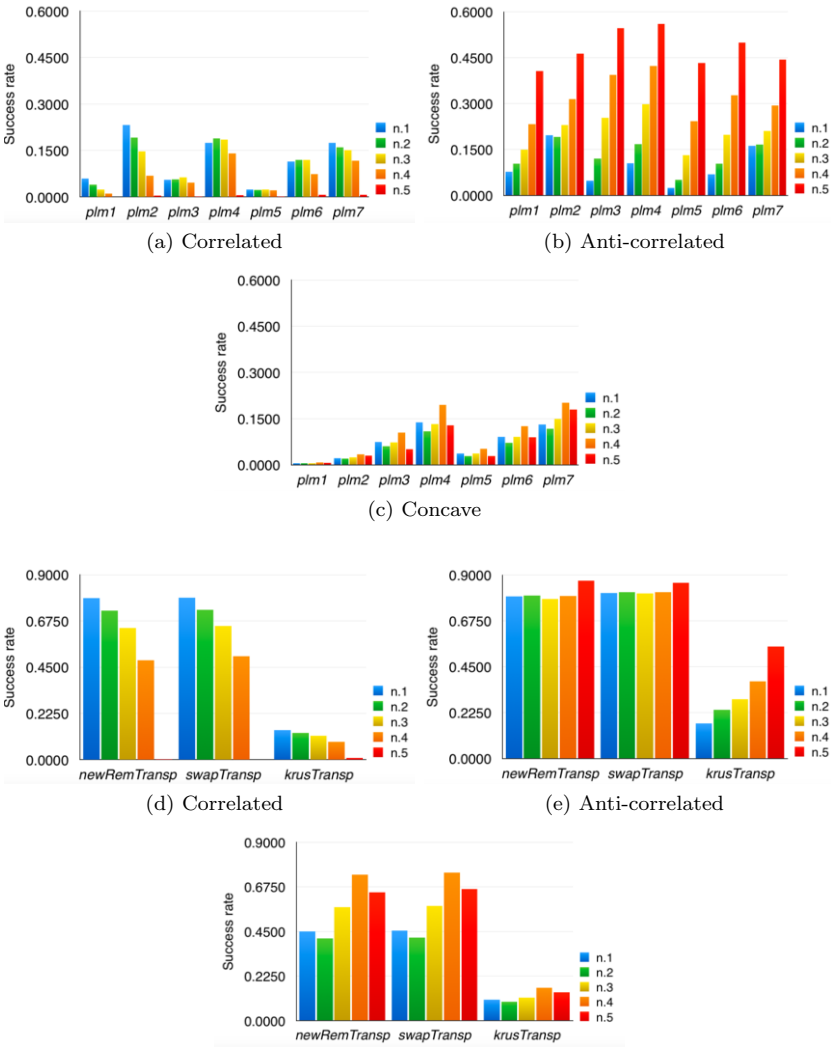

Fig. 1. Success rates of plasmids (a-c) and transposons (d-f) on KNW instances

by the number of successful manipulations divided by the total number of manipulations performed by that transgenetic agent. In average, for the correlated and concave instances, the plasmids that use the $m2$ manipulation method, e.g., *plm4* and *plm7*, contributed more to the search. The *plm2* also made a significant contribution to the correlated instances. All plasmids contributed to the search concerning the anti-correlated class. The *newRemTransp* and *swapTransp* presented similar success rates and were the most effective.

5 Conclusion

This study presented a transgenetic algorithm for the *BiST*, *TMA*, which deals with 10 transgenetic agents. The decision about which agent is selected to search around

the region of a given solution is made automatically and based on successful applications of the specific agents. The *TMA* was compared to an effective transgenetic algorithm in which the selection of the transgenetic agents depend on user-defined parameters and to the *NSGA-II*. The computational experiment comprised 165 benchmark instances divided into three classes. The analysis of the solutions produced by each algorithm was based on two quality indicators that measure different features of the approximation sets and statistical tests. We also analyzed the processing times. The results of the algorithm proposed in this study were superior both regarding solution quality and processing time. In future works, the algorithm will be extended to many-objective problems.

References

[1] Aggarwal, V., Y. Aneja and K. Nair, *Minimal spanning tree subject to a side constraint*, Computers & Operations Research, **9** (1982), 287–296.

[2] Arroyo, J. E. C., P. S. Vieira and , D. S. Vianna, *A GRASP algorithm for the multi-criteria minimum spanning tree problem*, Annals of Operations Research, **159** (2008), 125–133.

[3] Bader, J., and E. Zitzler. *Hype: An algorithm for fast hypervolume-based many-objective optimization*, Evolutionary Computation, **19** (2011), 45–76.

[4] Battiti, R., and G. Tecchiolli. *The reactive tabu search*, ORSA Iournal on Computing, **6(2)** (1994), 126–140.

[5] Chen, G., S. Chen, W. Guo, and H. Chen, *The multi-criteria minimum spanning tree problem based genetic algorithm*, Information Sciences, **117** (2007), 5050–5063.

[6] Coello, C. A. and M. R. Sierra, *A study of the parallelization of a coevolutionary multi-objective evolutionary algorithm, in: Mexican International Conference on Artificial Intelligence*, Springer 2004, 688–697.

[7] Corley, H. W., *Efficient spanning trees* Journal of Optimization Theory and Applications, **45** (1985), 481-485

[8] Deb, K., A. Pratap, S. Agarwal, and T. Meyarivan, *A fast and elitist multiobjective genetic algorithm: NSGA-II*, IEEE Transactions on Evolutionary Computation, **6** (2002), 182–197.

[9] Davis-Moradkhan, M., and W. N. Browne, *A knowledge-based evolution strategy for the multi-objective minimum spanning tree problem, in: IEEE Congress on Evolutionary Computation*, CEC 2006, 1391–1398.

[10] Ehrgott, M., and X. Gandibleux, *A survey and annotated bibliography of multiobjective combinatorial optimization*, OR Spektrum, **22** (2000) 425–460.

[11] Friedman, M., *The use of ranks to avoid the assumption of normality implicit in the analysis of variance*, Journal of the American Statistical Association, **32** (1937), 675–701.

[12] Goldbarg, E. F. G., M. C. Goldbarg, *A new endosymbiotic approach for evolutionary algorithms*, Foundations of Computational Intelligence, **3** (2009), 425–460.

[13] Hamacher, H. W., and G. Ruhe, *On spanning tree problems with multiple objectives*, Annals of Operations Research, **52** (1994), 209–230.

[14] Holm, S., *A simple sequentially rejective multiple test procedure*, Scandinavian Journal of Statistics **6** (1979), 65–70.

[15] Knowles, J. D.,"Local-search and hybrid evolutionary algorithms for Pareto optimization," Ph.D. thesis Department of Computer Science, University of Reading, Reading, UK, 2002.

[16] Knowles, J., L. Thiele and E. Zitzler, "A Tutorial on the Performance Assessment of Stochastic Multiobjective Optimizers," Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, 2005.

[17] Kruskal, J. B., *On the shortest spanning subtree of a graph and the traveling salesman problem*, Proceedings of the American Mathematical Society, **7** (1956), 48–50.

[18] López-Ibáñez, M., J. Dubois-Lacoste, L.P. Cáceres, M. Birattari and T. Sttzle, *The irace package: Iterated racing for automatic algorithm configuration*, Operations Research Perspectives, **3** 2016, 43–58.

[19] J. Párraga-Álava, M. Dorn and M. Inostroza-Ponta, *Using local search strategies to improve the performance of NSGA-II for the Multi-Criteria Minimum Spanning Tree problem, in: IEEE Congress on Evolutionary Computation*, CEC 2017, 1119–1126.

[20] Monteiro, S. M. D., E. F. G. Goldbarg and M. C. Goldbarg, *A new transgenetic approach for the biobjective spanning tree problem, in: Proceedings of IEEE Congress on Evolutionary Computation*, IEEE CEC 2010, **1**, 519–526.

[21] Prim, R. C., *Shortest connection networks and some generalizations*, Bell System Technical Journal, **36** (1957), 1389–1401.

[22] Pugliese, L. P., F. Guerriero, and J. L. Santos, *Dynamic programming for spanning tree problems: application to the multi-objective case* Optimization Letters, **9** (2015), 437–450

[23] Raidl G. R., and B. A. Julstrom, *Edge sets: an effective evolutionary coding of spanning trees*, IEEE Transactions on Evolutionary Computation, **7** (2003), 225–239.

[24] Ramos, R. M., S. Alonso, J. Sicilia and C. González, *The problem of the optimal biobjective spanning tree* European Journal of Operational Research, **111** (1998), 617–628

[25] Rocha, D. A. M., E. F. G. Goldbarg and M. C. Goldbarg, *A memetic algorithm for the biobjective minimum spanning tree problem, in: European Conference on Evolutionary Computation in Combinatorial Optimization*, EvoCOP 2006, 222–233.

[26] Rocha, D. A. M., E. F. G. Goldbarg and M. C. Goldbarg, *A new evolutionary algorithm for the bi-objective minimum spanning tree, in: Proceedings of Seventh International Conference on Intelligent Systems Design and Applications*, ISDA 2007, 735–740 .

[27] Ruzika, S., and H. W. Hamacher, "A survey on multiple objective minimum spanning tree problems," Algorithmics of Large and Complex Networks Design, Analysis, and Simulation Wagner, J. L. D. and Zweig, K. A., 104–116, 2009.

[28] Seada, H., and Deb, K. *A unified evolutionary optimization procedure for single, multiple, and many objectives*, IEEE Transactions on Evolutionary Computation, **20(3)** (2016), 358–369.

[29] Sourd, F., and O. Spanjaard, *A multiobjective branch-and-bound framework: Application to the biobjective spanning tree problem*, INFORMS Journal on Computing, **20** (2008), 472–484.

[30] Steiner, S., and T. Radzik, *Computing all efficient solutions of the biobjective minimum spanning tree problem* Computers & Operations Research, **35** (2008), 198–211

[31] Zhou, G. and M. Gen, *Genetic algorithm approach on multi-criteria minimum spanning tree problem*, European Journal of Operational Research, **114** (1999), 141–152.

[32] Zitzler, E. and L. Thiele, *Multiobjective optimization using evolutionary algorithms a comparative case study, in: Parallel Problem Solving from Nature PPSN V: 5th International Conference*, PPSN V 1998, **1498**, 292–301.