Fixed Parameter Evolutionary Algorithms and Maximum Leaf Spanning Trees: A Matter of Mutation

Stefan Kratsch[1], Per Kristian Lehre[2,3],
Frank Neumann[1], Pietro Simone Oliveto[3]

[1] Algorithms and Complexity, Max-Planck-Institut für Informatik,
Saarbrücken, Germany
[2] DTU Informatics, Technical University of Denmark, 2800 Kgs. Lyngby, Denmark
[3] School of Computer Science, University of Birmingham,
Birmingham, United Kingdom

Abstract. Evolutionary algorithms have been shown to be very successful for a wide range of NP-hard combinatorial optimization problems. We investigate the NP-hard problem of computing a spanning tree that has a maximal number of leaves by evolutionary algorithms in the context of fixed parameter tractability (FPT) where the maximum number of leaves is the parameter under consideration. Our results show that simple evolutionary algorithms working with an edge-set encoding are confronted with local optima whose size of the inferior neighborhood grows with the value of an optimal solution. Investigating two common mutation operators, we show that an operator related to spanning tree problems leads to an FPT running time in contrast to a general mutation operator that does not have this property.

1 Introduction

Evolutionary Algorithms (EAs) are a large class of stochastic search algorithms that are widely used to solve combinatorial optimization problems. They have found many applications for different kinds of NP-hard spanning tree problems (see e. g. [10, 5]). Our aim is to contribute to the theoretical understanding of evolutionary algorithms for such kind of problems. Rigorous runtime analyses have been widely used to provide theoretical insights into the optimization process of evolutionary algorithms and we follow this line of research throughout this paper. The first runtime analyses of EAs were performed on artificially created pseudo-Boolean functions to understand what characteristics of a problem make its optimisation easy or hard for an EA (see e. g. [2]). These first efforts led to the development of a range of mathematical techniques used for the analyses. Building up on these results it has been possible to analyse the performance of EAs on classical combinatorial optimisation problems (see [9] for an overview).

Recently, the notion of fixed parameter tractability has been introduced into the theoretical analysis of evolutionary algorithms [6]. A parameterized analysis

allows a more detailed inspection on which instances of an NP-hard combinatorial optimization problem are hard to solve. Such an analysis depends on a parameter k which measures the difficulty of the problem under consideration. In parameterized complexity, a problem with parameter k is *fixed parameter tractable* (FPT) if there exists an algorithm that decides it in time $O(f(k) \cdot n^c)$ [1]. Hence, the runtime of the *FPT algorithm* is $O(n^c)$ for every fixed value of k. For many real-world instances of NP-hard problems, the parameter k in question is bounded, and not too large. Hence, these problem instances can solved by FPT-algorithms in polynomial time, despite the general problem being NP-hard. We point out that problems considered in parameterized complexity often have straightforward $O(n^{f'(k)})$ time algorithms; however, while also polynomial for every fixed k, the degree of the polynomial does depend on k. Fixed-parameter evolutionary algorithms are evolutionary algorithms that compute an optimal solution in expected time $O(f(k) \cdot n^c)$. In [6], it has been shown that there are fixed parameter evolutionary algorithms for the vertex cover problem.

We put forward the parameterized analysis of evolutionary algorithms and investigate this kind of algorithms for the computation of a maximum leaf spanning tree. There are different approximation algorithms based on local search for this problem that give a constant approximation ratio [8, 7]. On the other hand, it is known that the problem is APX-complete [4]. We consider exact optimization and investigate two evolutionary algorithms working with an edge-set encoding [10] which is very popular when solving spanning tree problems. Our algorithms differ from each other by the chosen mutation operator. The more general mutation operator is motivated by standard bit-mutation and does not necessarily create a tree whereas the second (more problem-specific) operator makes sure that each created offspring is a tree. We present instances containing a local optima that is hard to leave if the value of an optimal solution is large. Based on the structure of these instances, we prove lower bounds on the expected optimization time for both algorithms which grow with the value of an optimal solution. Later on, we show that the more problem specific mutation operator leads to fixed parameter evolutionary algorithms for the maximum leaf spanning tree problem, while the more general mutation operator does not have this property according to our proven lower bounds.

After having motivated our work, we introduce the problem and algorithms in Section 2. We present an instance with a local optimum and a large inferior neighborhood in Section 3. In Section 4, we show that a suitable mutation operator leads to fixed parameter evolutionary algorithms for the maximum leaf spanning tree problem. Finally, we finish with some concluding remarks.

2 Problem and Algorithms

We investigate the following NP-hard spanning tree problem. Given an undirected connected graph $G = (V, E)$, the goal is to find a spanning tree T^* of G such that the number of leaves is maximal.

We consider two simple evolutionary algorithms which differ by the choice of the mutation operator. Both algorithms start with an arbitrary spanning tree T of G. We denote by m the number of edges in G, and $\ell(T)$ the number of leaves of the spanning tree T. A new solution is only accepted if it is a spanning tree whose number of leaves is at least as high as the number of leaves in the current solution. The first algorithm can be described as follows.

Algorithm 1 (Generic (1+1) EA)

1. *Choose a spanning tree of T uniformly at random.*
2. *Produce T' by swapping each edge of T independently with probability $1/m$.*
3. *If T' is a tree and $\ell(T') \geq \ell(T)$, set $T := T'$.*
4. *Go to 2.*

Swapping an edge in step 2. of Algorithm 1 means that if an edge is present in T then it is not contained in T' with probability $1/m$. On the other hand, if an edge is not present in T then it is contained in T' with probability $1/m$. An edge does not change from T to T' with probability $1-1/m$ in each mutation step independently of the other edges. Note, that the mutation operator of Algorithm 1 does not necessarily create an offspring that is a tree. If the offspring is not a tree then this individual is discarded as it represents an infeasible solution.

Often it is assumed that choosing a mutation operator that is more tailored to the problem gives a significant speed up. The second algorithm uses a problem-specific mutation operator that ensures valid solutions, i. e. spanning trees.

Algorithm 2 (Tree-Based (1+1) EA)

1. *Choose an arbitrary spanning tree T of G.*
2. *Choose S according to a Poisson distribution with parameter $\lambda = 1$ and perform sequentially S random edge-exchange operations to obtain a spanning tree T'. A random exchange operation applied to a spanning tree $\tilde{T}$ chooses an edge $e \in E \setminus \tilde{T}$ uniformly at random. The edge e is inserted and one randomly chosen edge of the cycle in $\tilde{T} \cup \{e\}$ is deleted.*
3. *If $\ell(T') \geq \ell(T)$, set $T := T'$.*
4. *Go to 2.*

Our goal is to point out the differences between the two algorithms. To do this, we compare the expected number of iterations that our algorithms need to compute an optimal solution. The expected number of iterations needed to obtain an optimal solution is called the *expected optimization time*, and is the commonly used performance measure in the rigorous runtime analysis of evolutionary algorithms. We will show that choosing the more problem-specific mutation operator of Algorithm 2 makes the difference between a fixed-parameter evolutionary algorithm and an evolutionary algorithm that does not compute an optimal solution within expected FPT-time.

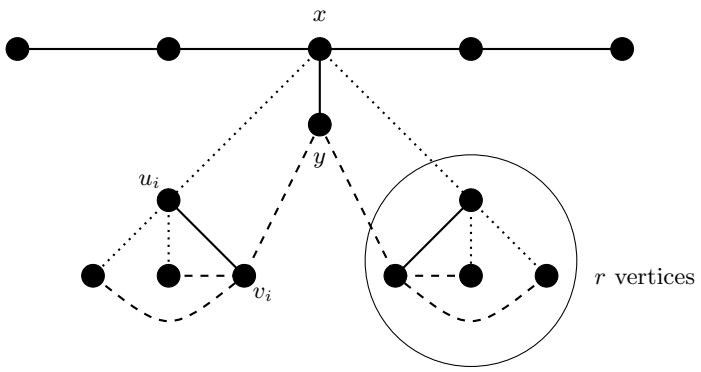

Fig. 1. Local optimum shown with dashed edges, global optimum with dotted edges, shared edges are drawn solid.

3 Local Optima and Lower Bounds

The aim of this section is to point out structures of the problem that make it hard for our algorithms to achieve an improvement. We discuss the presence of local optima and present a graph that consists of a local optimum which has a large distance (in terms of the number of edge exchanges) from the global optimum. Using this observation, we show lower bounds on the expected optimization time for the two algorithms under consideration.

Our graph called G_{loc} (see Figure 1) contains two components consisting of r vertices each. In component i, $1 \leq i \leq 2$, two vertices u_i and v_i are connected to all the other vertices in that component. The vertex u_i is connected to vertex x which lies outside the component. Similarly vertex v_i is connected to vertex y. In addition, x and y share an edge. The graph is completed by attaching a path of $n-2r-2$ vertices to the vertex x. A tree has to contain all the edges of the path attached to x. For a given component, the maximal number of possible leaves is at most $r-1$. This can be obtained by attaching all nodes of the component either to u_i or v_i.

The graph contains a local optimum T_{lopt} which consists of all edges attached to the vertices v_i, $1 \leq i \leq 2$, the edge $\{x, y\}$ and all path edges. The global optimum T_{opt} consists of all edges attached to the vertices u_i, $1 \leq i \leq 2$, the edge $\{x, y\}$ and all path edges. Compared to T_{lopt}, T_{opt} has an extra leaf, namely the vertex y. However, T_{lopt} and T_{opt} differ by $4(r-1)$ edges which make it hard for the algorithms under consideration to obtain T_{opt} if T_{lopt} has been produced before.

Our goal is to study the expected optimization time of the algorithms introduced in the previous section in dependence of the number of leaves which, in turn, depends on r. To do this, we first consider the number of different spanning trees of G_{loc} in dependence of r.

Lemma 1. *The number of spanning trees of G_{loc} is at most 2^{4r}.*

Proof. A spanning tree has to contain all edges of the path attached to x. The path attached to x consists of $n-2r-2$ edges. A spanning tree contains exactly $n' = n-1-(n-2r-2) = 2r+1$ non-path edges.

We count the total number of non-path edges in G_{loc}. Consider a component consisting of r edges. The number of edges within such a component is $2r-3$ as u_i and v_i are connected to all other vertices and share an edge. In addition there are two edges connecting each component to the outer part. Hence, the total number of edges connected to vertices of a single component is $2r-1$. In addition, there is the edge connecting x and y.

Summing up, the graph consists of $m' = 2(2r-1)+1 = 4r-1$ non-path edges. The number of different spanning trees is therefore at most

$$\binom{m'}{n'} = \binom{4r-1}{2r+1} \leq 2^{4r}.$$

□

Using the previous lemma, we show the following lower bound on the expected optimization time of Generic (1+1) EA on G_{loc}.

Theorem 1. *The expected optimization time of Generic (1+1) EA on G_{loc} is lower bounded by $\left(\frac{m}{c}\right)^{2(r-2)}$ where c is an appropriate constant.*

Proof. The number of spanning trees of G_{loc} is at most 2^{4r}. Therefore, the initial spanning tree is T_{lopt} with probability at least 2^{-4r}. This spanning tree is a local optimum with $2(r-1)+2$ leaves. In order to obtain a different spanning tree with at least as many leaves, $r-1$ leaves have to be achieved in each component, or at least $r-1$ leaves have to be obtained in one component and y has to become a leaf. Hence, in order to achieve an accepted solution that is different from T_{lopt} all $(r-2)$ nodes of at least one component i have to be assigned to u_i instead of v_i. This implies that at least $2(r-2)$ edges for a fixed component have to be swapped to escape from the local optimum. There are two components where this can happen which implies that the probability for such a step is at most $2\left(\frac{1}{m}\right)^{2(r-2)}$. The expected waiting time for such a step is at least $\frac{1}{2} \cdot m^{2(r-2)}$. Altogether the expected optimization time is lower bounded by

$$2^{-4r} \cdot \frac{1}{2} \cdot m^{2(r-2)} \geq \left(\frac{m}{c}\right)^{2(r-2)},$$

where c is an appropriate constant. □

Using the previous ideas, we can also lower bound the expected optimization time of Tree-Based (1+1) EA on G_{loc}.

Theorem 2. *The expected optimization time of Tree-Based (1+1) EA on G_{loc} is lower bounded by $(\frac{r-2}{c})^{r-2}$ where c is an appropriate constant.*

Proof. We follow the ideas of the previous theorem. With probability at least 2^{-4r}, T_{lopt} is chosen as the initial spanning tree. In order to produce from T_{lopt} the optimal solution T_{opt}, $(r-2)$ exchange operations have to be carried out in a single mutation step. According to the Poisson distribution with $\lambda = 1$, the probability that this happens in the next step is

$$\frac{1}{e(r-2)!} \leq \frac{1}{\sqrt{2\pi(r-2)}} e^{r-3}(r-2)^{-(r-2)} \leq e^{r-3}(r-2)^{-(r-2)}.$$

Altogether the expected optimization time is lower bounded by

$$2^{-4r} \cdot e^{-r+3}(r-2)^{(r-2)} \geq \left(\frac{r-2}{c}\right)^{r-2},$$

where c is an appropriate constant. □

To show that both algorithms need not only in expectation that many steps, but also with a high probability the graph can be modified such that it consists of more than two components attached to x and y. Then a typical run can be investigated to show that at least two components end up in the local optimum.

4 FPT of Edge Exchanges

In this section we prove that Algorithm 2 is an FPT algorithm for the maximum leaf spanning tree problem with respect to the maximal number of leaves k. Given that the maximal-leaf spanning tree has k leaves, in the following lemma we derive upper bounds in dependence of k on the number of edges and on the number of nodes of degree at least three that the graph may contain. These bounds will allow us to prove the main result of this section presented in Theorem 3.

The lemma is proven using an approach similar (but greatly simplified) to the one used in [3]; our focus here is on giving a self-contained presentation sufficient for obtaining the claimed expected runtime. Note also, that kernelization results, such as [3], almost always require a modification of the problem instance while we are interested in bounding the original instance.

Lemma 2. *Any connected graph G on n nodes and with a maximum number of k leaves in any spanning tree has at most $n+5k^2-7k$ edges and at most $10k-14$ nodes of degree at least three.*

Proof. Let G be a graph on n nodes and let T be a spanning tree of G with (the maximum number of) k leaves. We let P_0 denote the set of all leaves and all nodes of degree at least three in T. (We denote the degree of node x within the tree T by $\deg_T(x)$.) Furthermore, let $P \supseteq P_0$ denote the set of all nodes that are within distance of at most two of any node of P_0 (distance and degree w.r.t. T). We let Q denote the set of remaining nodes.

In the following we show that all nodes of Q have degree two in G (clearly they have degree at least two in G since they have degree two in T). We assume for contradiction that there is a node $v \in Q$ which has degree at least three in G. Therefore, v has a neighbor u in G to which it is not adjacent in T. We distinguish two cases:

If u is at distance two from v (w.r.t. T) then it is neither a leaf nor does it have degree greater than two. Let w be the node that is adjacent to both u and v in T. Observe that adding the edge $\{u, v\}$ to T and removing $\{v, w\}$ creates a spanning tree with an additional leaf, namely w (note that the graph does not disconnect since we remove an edge of a cycle). This contradicts our choice of T.

If u is at distance greater than two from v (w.r.t. T) then we observe the following: Adding the edge $\{u, v\}$ to T creates a cycle $C = \{\ldots, u, v, w, x, \ldots\}$ on at least four nodes (else u would be too close to v). Since $v \in Q$ both nodes w and x (to which v is at distance at most two in T) have degree two in T. Thus adding the edge $\{u, v\}$ to T and removing $\{w, x\}$ would give two additional leaves w and x while possibly losing the leaf u (again, removing an edge from a cycle must give a connected graph). This contradicts our choice of T.

Thus all nodes in Q have degree two in G. We now bound the size of P. To this end, we make the following observations: T has k leaves and thus it has at most $k-2$ nodes of degree at least three. Also, the number of leaves in any tree is equal to 2 plus $deg(v) - 2$ for every node v of degree at least three, so

$$\sum_{v:deg_T(v)\geq 3} (deg_T(v) - 2) = k - 2.$$

The number of elements in P_0 are the k leaves, plus the at most $k-2$ nodes of degree at least 3. Let P_1 be the set of nodes at distance 1 from P_0, and P_2 the set of nodes at distance 2 from P_0. The number of nodes in P_1 that are connected to a leaf node in P_0 is at most k. The number of nodes in P_1 that are connected to a node of degree at least 3 in P_0 is at most

$$\begin{aligned}\sum_{v:deg_T(v)\geq 3} deg_T(v) &= \sum_{v:deg_T(v)\geq 3} (deg_T(v) - 2 + 2) \\ &\leq 2(k-2) + \sum_{v:deg_T(v)\geq 3} (deg_T(v) - 2) \leq 3k - 6.\end{aligned}$$

In total, there are no more than $4k - 6 = k + (3k - 6)$ nodes in P_1. Finally, each node in P_1 has degree two and is adjacent to at least one node of P_0. Furthermore, each node of P_2 is adjacent to at least one node of P_1. Therefore $|P_2| \leq |P_1|$. In total, there are no more than $|P_0| + |P_1| + |P_2| \leq 10k - 14$ nodes in P. Clearly, $10k - 14$ is also an upper bound on the number of nodes of degree at least three in G since they cannot be contained in Q.

To bound the number of edges we observe that no node of G can have degree greater than k: Starting a tree from a node of degree greater than $k + 1$ and adding the remaining nodes to this tree would give a spanning tree with more

than k leaves. Thus we get the claimed upper bound on the number m of edges:

$$m \leq \frac{1}{2}(k|P| + 2|Q|) = \frac{k}{2}|P| + |Q| \leq 5k^2 - 7k + n.$$

This completes the proof. □

Now we are ready to prove the main result. Since a spanning tree always has $n-1$ edges, from Lemma 2 there are at most $5k^2$ edges to choose from at each step and at most all of them need to be replaced to reach the optimal spanning tree. The proof of the following theorem first shows that the probability of increasing the number of leaves by one in the current (non-optimal) spanning tree only decreases with the fixed parameter k. The proof is concluded by showing that the probability of exchanging all the $5k^2$ edges in one mutation step also depends only on k leading to the claimed runtime.

Theorem 3. *If the maximal number of leaf nodes in any spanning tree of G is k, then Algorithm 2 finds an optimal solution in expected time $O(2^{15k^2 \log k})$.*

Proof. Let $n_{\geq 3}$ be the number of nodes with degree at least three. We call an edge *distinguished* if it is incident on a node of degree at least 3, and non-distinguished otherwise. By applying Lemma 2, the number of distinguished edges on any cycle is at most $2n_{\geq 3} \leq 20k - 28$, since there are at most $n_{\geq 3}$ nodes of degree at least 3 on the cycle, and each node is incident with at most two edges of the cycle.

We first bound the probability of reducing the distance to an optimal spanning tree by 1. Let $E^* \subseteq E$ be the optimal spanning tree that is closest to the current spanning tree, and let e be any edge in E^* that is not yet in the current spanning tree. By Lemma 2, the number of edges in the graph is $m \leq n+5k^2-7k$. So the probability that edge e is introduced in an edge exchange operation is at least $1/(m-(n-1)) \geq 1/5k^2$. Introducing edge e creates a cycle. Consider first the case when the cycle consists only of distinguished edges. The length of such a cycle is no more than $20k - 28$, and the probability of removing one of the edges that is not in the optimal spanning tree is at least $1/20k$. In the case where the cycle contains non-distinguished edges, we claim that it suffices to remove any non-distinguished edge e' from the cycle. The claim obviously holds when the chosen edge e' is not in the optimal spanning tree, so assume that edge e' is in the optimal spanning tree. A *bridge edge* in a connected graph is any edge e such that the subgraph on the edges $E \setminus \{e\}$ is disconnected. Edge e' connects two components T_1 and T_2 in E^*, and cannot be a bridge edge because then the edge could not have been part of a cycle. Since edge e connects T_1 and T_2, the cycle must contain at least one other edge e'' that connects T_1 and T_2, and this edge is not part of the optimal spanning tree E^*. However, the spanning tree $(E^* \setminus \{e'\}) \cup \{e''\}$ must also be optimal, because adding edge e'' decreases the number of leaf nodes by at most 2, and removing edge e' increases the number of leaf nodes by exactly 2. Hence, adding edge e and removing edge e' reduces the distance to an optimal spanning tree by 1. Let ℓ be the number of non-distinguished edges on the cycle. No cycle contains more than $20k - 28$ distinguished edges, so the probability of removing a non-distinguished edge is

at least $\ell/(20k - 28 + \ell) \geq 1/20k$. The probability of reducing the distance to a global optimum by 1 is therefore at least $1/(20k \cdot 5k^2)$.

The number of edges r that must be inserted in the spanning tree is no more than $m-(n-1) \leq 5k^2$. The edges can be inserted in any order. The probability that in Step 2 of the algorithm, we choose to do $S = r$ operations is $1/er!$. So, the probability that in one step, we decide to do r edge exchange operations in any of the $r!$ orders, and each of the edge exchanges decreases the Hamming distance to an optimal spanning tree is at least

$$r! \cdot \frac{1}{er!} \cdot \left(\frac{1}{5k^2} \cdot \frac{1}{20k}\right)^r \geq \frac{1}{e}\left(\frac{1}{100k^3}\right)^{5k^2} \geq \frac{1}{e}\left(\frac{1}{100}\right)^{5k^2}\left(\frac{1}{k}\right)^{3\cdot 5k^2},$$

which implies that the expected number of steps to find an optimal spanning tree is at most $O(2^{15k^2 \log k})$. □

Conclusions

The parameterized complexity analysis of evolutionary algorithms is a promising research direction that is likely to become an important part in the theoretical analysis of evolutionary computation during the next years. An advantage in comparison to classical worst-case considerations is that this kind of analysis gives characterizations of what difficult instances for a specific algorithm look like in relation to some parameter of the problem. Evolutionary algorithms have produced very good results for different kind of NP-hard spanning tree problems. In this paper, we have studied evolutionary algorithms for the NP-hard maximum leaf spanning tree problem in the context of parameterized complexity. In our case the parameter is the size of the global optimum. Our investigations show that there may be local optima where the size of an inferior neighborhood grows with the number of leaves in optimal solutions. Investigations of two common mutation operators point out that a more problem-specific operator makes the difference between a fixed parameter evolutionary algorithm for the maximum leaf problem and an algorithm that does not have this property.

Acknowledgements Per Kristian Lehre was supported by EPSRC under grant no. EP/D052785/1, and Deutsche Forschungsgemeinschaft (DFG) under grant no. WI 3552/1-1. Pietro Simone Oliveto was supported by EPSRC under grant no. EP/P502322/1.

References

1. R. G. Downey and M. R. Fellows. *Parameterized Complexity (Monographs in Computer Science)*. Springer, November 1998.
2. S. Droste, T. Jansen, and I. Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276:51–81, 2002.

3. M. R. Fellows, D. Lokshtanov, N. Misra, M. Mnich, F. A. Rosamond, and S. Saurabh. The complexity ecology of parameters: An illustration using bounded max leaf number. *Theory Comput. Syst.*, 45(4):822–848, 2009.
4. G. Galbiati, F. Maffioli, and A. Morzenti. A short note on the approximability of the maximum leaves spanning tree problem. *Inf. Process. Lett.*, 52(1):45–49, 1994.
5. J. D. Knowles and D. Corne. A comparison of encodings and algorithms for multi-objective spanning tree problems. In *Proceedings of the Congress on Evolutionary Computation 2001*, pages 544–551. IEEE Press, 2001.
6. S. Kratsch and F. Neumann. Fixed-parameter evolutionary algorithms and the vertex cover problem. In F. Rothlauf, editor, *GECCO*, pages 293–300. ACM, 2009.
7. H.-I. Lu and R. Ravi. Approximating maximum leaf spanning trees in almost linear time. *J. Algorithms*, 29(1):132–141, 1998.
8. H.-I. Lu, R. Ravi, and R. Ravi. The power of local optimization: Approximation algorithms for maximum-leaf spanning tree. In *In Proceedings, Thirtieth Annual Allerton Conference on Communication, Control and Computing*, pages 533–542, 1996.
9. P. S. Oliveto, J. He, and X. Yao. Time complexity of evolutionary algorithms for combinatorial optimization: A decade of results. *International Journal of Automation and Computing*, 4(3):281–293, 2007.
10. G. R. Raidl and B. A. Julstrom. Edge sets: an effective evolutionary coding of spanning trees. *IEEE Trans. Evolutionary Computation*, 7(3):225–239, 2003.