An Optimal Algorithm to Find Champions of Tournament Graphs*

Lorenzo Beretta[1], Franco Maria Nardini[2], Roberto Trani[2,3], and Rossano Venturini[2,3]

[1] Scuola Normale Superiore, Pisa, Italy
lorenzo.beretta@sns.it
[2] ISTI-CNR, Pisa, Italy
{francomaria.nardini,roberto.trani,rossano.venturini}@isti.cnr.it
[3] Department of Computer Science, University of Pisa, Italy
{roberto.trani,rossano.venturini}@unipi.it

Abstract. A tournament graph $T = (V, E)$ is an oriented complete graph, which can be used to model a round-robin tournament between n players. In this short paper, we address the problem of finding a champion of the tournament, also known as Copeland winner, which is a player that wins the highest number of matches. Our goal is to solve the problem by minimizing the number of *arc lookup*s, i.e., the number of matches played. We prove that finding a champion requires $\Omega(\ell n)$ comparisons, where ℓ is the number of matches lost by the champion, and we present a deterministic algorithm matching this lower bound without knowing ℓ. Solving this problem has important implications on several Information Retrieval applications including Web search, conversational AI, machine translation, question answering, recommender systems, etc.

Keywords: Tournament Graph · Round-Robin Tournament · Copeland Winner

1 Introduction

A *tournament graph* is an oriented complete graph $T = (V, E)$ [13]. This graph can be used to model a round-robin tournament between n players, where each player plays a match with any other player. The orientation of an arc in E tells the winner of the match, i.e., we have the arc $(u, v) \in E$ iff u beats v in their match. In this short paper, we address the problem of finding a champion of the tournament, also known as *Copeland winner* [6], which is a vertex in V with the maximum out-degree, i.e., a player that wins the highest number of matches. Our goal is to find a champion by minimizing the number of *arc lookup*s, i.e., the number of played matches. Note that a tournament graph may have more

* This paper is partially supported by the BIGDATAGRAPES (EU H2020 RIA, grant agreement N°780751), the "Algorithms, Data Structures and Combinatorics for Machine Learning" (MIUR-PRIN 2017), the OK-INSAID (MIUR-PON 2018, grant agreement N°ARS01_00917) projects, and Scuola Normale Superiore.

than one champion. In this case, we aim at finding any of them, even if all the algorithms are able to find all of them without increasing the complexity.

If the tournament is transitive (i.e., if u wins against v and v wins against w, then u wins against w), we can trivially identifying the (unique) champion with $\Theta(n)$ arc lookups. Indeed, the champion is the only vertex that wins all its matches and, thus, we can perform a *knock-out tournament* where the loser of any match is immediately eliminated. However, finding the champion of general tournament graphs requires $\Omega(n^2)$ arc lookups [7] and, thus, there is nothing better to do than playing all the matches. This means that the structure of the underlying tournament graph heavily impacts the complexity of the problem.

In this paper we parametrize the problem with the number ℓ of matches lost by the champion. We first show that $\Omega(\ell n)$ arc lookups are required, then we present an optimal deterministic (and non-trivial) algorithm for finding a champion and achieving this bound without knowing ℓ. This parametrization is motivated by applications in Information Retrieval and Machine Learning where we expect that a champion (e.g., the best item in a set) loses only few matches.

Motivations. The identification of the best candidate of a set of items is a crucial task in many Information Retrieval (IR) applications, also known as P@1, including Web search, conversational AI, machine translation, question answering, recommender systems, etc. [11,8]. The task can be solved in two different ways: i) by piggybacking state-of-the-art Machine Learning (ML) techniques for ranking, and by selecting the candidate with the highest rank in the list; ii) by employing a pairwise ML classifier, which is trained to identify the best among two candidates, and then by selecting the champion of the resulting tournament. While the former approach exploits only the information of a single candidate for computing the ranking, the latter approach is potentially more powerful because it exploits the information of two candidates while comparing pairs of items. However, the latter approach is more expensive than the former one due to the lack of efficient algorithms to reduce the number of time-consuming comparisons. This work aims at filling this gap since the P@1 problem can be solved by finding a champion of the tournament graph induced by the pairwise ML classifier. As it is possible to design very accurate ML classifiers for several tasks, we expect a very low number of matches ℓ lost by the best item, thus a quasi-linear number of arc lookups is required by our algorithm to find it.

2 Related Work

Tournament graphs are a well-known model that has been applied to several different areas such as sociology, psychology, statistics, and computer science. Examples of applications are round-robin tournaments, paired-comparison experiments, majority voting, communication networks, etc. [4,13,9,10,12].

For the purpose of this work, we identify two different research lines. The first one aims at defining different *notions of tournament winner*, while the second one aims at *efficiently ranking the list of candidates* using pairwise approaches.

Tournament winner. According to previous works [4,10,12], there is no unique definition of the notion of a tournament winner. Nevertheless, all of them agree on defining the winner whenever there is a candidate, called *Condorcet winner*, which beats all the others. Different definitions of winner requires different complexities of the algorithms used to identify it.

The easiest cases to consider are the *transitive tournament graphs*, i.e., directed acyclic graphs, where it is trivial to find the Condorcet winner in linear time by performing a knock-out tournament. Instead, in the general case, the complexity of finding a winner is usually much higher and strictly depends on the definition of winner. The winner as defined by Slater [15], called *Slater solution*, is the Condorcet winner in a modified tournament graph T'. T' is obtained by reversing the minimum number of arcs of T that make it a transitive tournament. However, the computation of the Slater solution is $\mathcal{NP}$-hard. The $\mathcal{NP}$-hardness derives by reduction to the *Feedback Arc Set Problem* [5]. A winner as defined by Banks [3] is the Condorcet winners of a maximal transitive sub-tournament of T. As there may be several of these sub-tournaments, the *Banks solution* is the set of all these winners. The problem of finding just one winner can be computed in $\Theta(n^2)$ arc lookups, while finding all of them is a $\mathcal{NP}$-hard problem [9]. A third definition of winner is given by Copeland [6], called *Copeland solution*, where a winner is the candidate winning the majority of the matches. This is the definition used in this paper. As we already mentioned, the Copeland solution requires $\Omega(n^2)$ arc lookups and there is a trivial algorithm to match it [7].

There are several other notions of winner, and most of them can be computed in polynomial time. We refer to Hudry [9] for a complete survey on this topic.

Tournament ranking. Several works deal with the problem of efficiently ranking vertices according to the structure of the tournament graph, see e.g., [13,9,14,2,1].

The result by Shen *et al.* [14] provides a ranking based on the definition of *king*. A vertex $u \in V$ is a king if for each $v \in V$ either 1) $(u, v) \in E$, or 2) $\exists w \in V$ such that $(u, w) \in E$ and $(w, v) \in E$. Equivalently, u is a king if for every vertex v there is a directed path from u to v of length at most 2 in T. Every tournament has at least a king and it can be easily computed in linear time. The ranking algorithm by Jian *et al.* [14] finds a sorted sequences of vertices $u_1, u_2, \ldots, u_n$ such that for every i 1) u_i beast u_{i+1}, and 2) u_i is a king in the sub-tournament induced by the items $u_{i+1}, u_{i+2}, \ldots, u_n$. The authors provide a $O(n^{3/2})$ algorithm to compute this sequence.

Ailon *et al.* [2,1] provide a bound to the error achieved by the Quicksort algorithm when used to sort vertices of the tournament graph. The error is defined as the number of mis-ordered pairs of vertices, i.e., $u, v \in V$ is such that u beats v, but v is ranked higher than u. Ailon *et al.* show that the expected error is at most two times the best possible error. It is apparent that the proposed algorithm requires $\Omega(n \log n)$ arc lookups with high probability.

We observe that the two results above are not suitable in our setting. Indeed, the definition of king is weaker than the one of Copeland winner, since the latter implies the former [13]. Moreover, it is easy to show that the algorithm by Ailon *et al.* fails in finding a winner w every time one of the Quicksort pivots beats w.

3 Algorithm

An adversarial argument is used by Gutin *et al.* [7] to prove that finding a champion requires $\Omega\left(n^2\right)$ arc lookups. Therefore, the trivial algorithm that finds a champion by performing all the possible matches is optimal in general. The problem is indeed much more interesting if we parameterize with ℓ, the number of matches lost by the champion. Note that ℓ is unknown to the algorithm.

The goal of this section is to prove the following theorem which states that $\Omega(\ell n)$ arc lookups are necessary to find a champion and to present an optimal algorithm requiring exactly that number of lookups.

Theorem 1. *Given a tournament graph T with n vertices and with ℓ matches lost by a champion, then*

- *finding a champion requires $\Omega(\ell n)$ arc lookups;*
- *there is an algorithm that finds every champion by requiring $\Theta(\ell n)$ arc lookups and time. The algorithm requires linear space.*

Lower bound. The lower bound is proved by using a simple adversarial argument. Assume that there is an algorithm that claims that a vertex u, losing ℓ matches, is a champion by performing $\frac{1}{2}\ell(n-1)$ arc lookups. There must exist a node v such that the algorithm has unfolded less than ℓ arcs incident to v. We thus can make v win more matches than u by setting v's unfolded matches, then let the algorithm be incorrect. In other words any correct algorithm, claiming that a vertex u is a champion with ℓ matches lost, must be able to certificate its answer exhibiting 1) a list of $n-1-\ell$ matches won by u, and 2) a list of ℓ matches lost by v, for every vertex $v \in V \setminus \{u\}$.

Upper Bound. To understand the difficulty of the problem, let us think about it in terms of the (unknown) adjacency matrix of the tournament graph. A reasonable strategy to solve the problem is by applying a row-wise approach: we process all the vertices, one after the other, and we try to discard each of them, say v, by finding at least ℓ matches lost by v, i.e., we try to certificate that v cannot be a champion. Through an adversarial argument, we can see that the best possible way to process any vertex v in this row-wise approach is to select random opponents until we are able to discard v (or recognize v as a champion). We can exploit properties of the global structure of the tournament to prove that this approach requires $\Theta\left(\ell n \log n\right)$ arc lookups with high probability. Thus, this algorithm is randomized and is $\log n$ times worse than the lower bound.

We design a simple, deterministic, and optimal algorithm to find the champion (Algorithm 1). The number ℓ of matches lost by the champion is unknown to the algorithm. Thus, it performs an exponential search to find the suitable value of α such that $\alpha/2 \leq \ell < \alpha$ (line 2) and tries to solve the problem by assuming that the champion loses less than α matches. At each iteration, the algorithm maintains a set A of alive vertices that is initially equal to V. Then, it performs an elimination tournament among the vertices in A by eliminating a player each

Algorithm 1

```
1: procedure FindChampion(T = (V, E))
2:     for (α = 1; true; α = 2α) do
3:         A = V
4:         S = {(u, u) | u ∈ V}
5:         ∀u ∈ V  lost[u] = 0    ▷ Initialize the number of lost matches of each vertex
6:         while |A| > 2α do
7:             choose a pair of vertices u, v in A^2 \ S
8:             S = S ∪ {(u, v), (v, u)}              ▷ Avoid selecting the same pair twice
9:             loser = if (u, v) ∈ E then v else u
10:            ++lost[loser]
11:            if lost[loser] ≥ α then
12:                A = A \ {loser}
13:        c, lost_c = FindChampionBruteForce(A, E)
14:        if lost_c < α then return c
```

time it loses α matches (line 12) until only 2α vertices remain alive (line 6). The matches are selected arbitrarily avoiding to play the same match multiple times (line 7). When the elimination tournament ends, a candidate champion is found via FindChampionBruteForce procedure, which exhaustively finds the vertex c of A with the maximum out-degree in T. Whenever the candidate c loses at least α matches (line 14), the value of α is not the correct one and the champion may have been erroneously eliminated before. Thus, c could be not a champion and the algorithm continues with the next value of α (line 2).

Correctness. Let us first assume that the value of α is such that $\alpha/2 \leq \ell < \alpha$. We prove now that, under this assumption, the algorithm correctly identifies a champion. First, we observe that the algorithm cannot eliminate the champions as any of them loses less than α matches. Thus, if we prove that the algorithm terminates, the set A contains all the champions and the FindChampionBruteForce procedure will identify any (potentially, all) of them. Note that a champion of T may not be a champion of the sub-tournament restricted to only the vertices in A. That is why FindChampionBruteForce procedure computes the out-degrees of all vertices in A by looking at the original tournament T. We use the following lemma to prove that eventually the condition $|A| = 2\alpha$ is met and the algorithm terminates.

Lemma 1. *In any tournament T of n vertices there is at least one vertex having in-degree $(n-1)/2$.*

Proof. The sum of the in-degrees of all vertices of T is exactly $\binom{n}{2} = \frac{n(n-1)}{2}$. Since there are n vertices, there must be at least one vertex with in-degree $\frac{n-1}{2}$.

Thus, each tournament of $2\alpha + 1$ vertices, or more, has at least one vertex losing at least α matches. This means that the algorithm has always the opportunity to eliminate a vertex from A until there are 2α vertices left. Notice that

the above discussion is valid for any value of α smaller than the target one. Thus, any iterations of the exponential search will terminate and it eventually finds a suitable value of α, i.e., $\alpha/2 \leq \ell < \alpha$, where a champion will be identified.

Complexity. Let us now analyze the complexity of the algorithm. Let us first consider the cost of an iteration of the exponential search. Observe that each arc lookup increases one entry of *lost* by one and that none of these entries is ever greater than α. Thus, the elimination tournament takes no more than $n\alpha$ arc lookups. Moreover, the FindChampionBruteForce procedure takes less than $2n\alpha$ arc lookups since it just unfolds every arc of the remaining 2α alive nodes. Thus, an iteration of the exponential search takes less than $3n\alpha$ arc lookups.

We get the complexity of the overall algorithm by summing up over all the possible values of α, which are all the powers of 2 from 1 up to 2ℓ. Thus, we have at most $3n \sum_{i=0}^{\lceil \log_2(2\ell) \rceil} 2^i = \Theta(\ell n)$ arc lookups.

Technical details. We are left with the proof that the Algorithm 1 can be implemented in $\Theta(\ell n)$ time and linear space. We do this by exploiting the fact that Algorithm 1 allows us to choose any arc as soon as its vertices are alive and it has never looked up before. An efficient implementation is achieved by maintaining two arrays of n elements each: an array A storing alive vertices, and an array *lost* storing the number of matches lost by each vertex. A counter *numAlive* stores the number of alive vertices. Our implementation maintains the invariant that the prefix $A[1, numAlive]$ contains only alive vertices. We use two cursors p_1 and p_2 to iterate over the elements in A. At the beginning $p_1 = 1$, $p_2 = 2$ and $numAlive = n$. Our implementation performs a series of matches involving vertex $A[p_1]$ and all other vertices in $A[p_1 + 1, numAlive]$. Then, it moves p_1 to the next position. After every match between $A[p_1]$ and $A[p_2]$, we increment *lost* of the loser, say vertex v. Whenever $lost[v]$ equals α we eliminate v according to the following two cases. The first case occurs when v is $A[p_1]$. We swap $A[p_1]$ and $A[numAlive]$, we end the current series of matches, and we start a new one. The second case occurs when v is $A[p_2]$. Here, we swap $A[p_2]$ and $A[numAlive]$, and we continue the current series of matches. In both cases, we decrease *numAlive* by 1 so that we preserve the invariant.

4 Conclusions and Future Work

We addressed the problem of finding champions in tournament graphs by minimizing the number of arc lookups. We showed that, given the number ℓ of matches lost by the champion, $\Omega(\ell n)$ arc lookups are required to find a champion. Then, we presented an optimal deterministic algorithm that solves the problem and matches the lower bound without knowing ℓ. As future work, we plan to experimentally evaluate the proposed algorithm to solve the Information Retrieval tasks outlined in the Introduction. We also plan to study the impact of randomization on this problem to design a Monte Carlo algorithm that lowers the complexity in charge of providing an incorrect output with small probability.

References

1. Ailon, N., Mohri, M.: An efficient reduction of ranking to classification. In: Servedio, R.A., Zhang, T. (eds.) 21st Annual Conference on Learning Theory - COLT 2008, Helsinki, Finland, July 9-12, 2008. pp. 87–98. Omnipress (2008), http://colt2008.cs.helsinki.fi/papers/32-Ailon.pdf
2. Ailon, N., Mohri, M.: Preference-based learning to rank. Machine Learning **80**(2-3), 189–211 (2010). https://doi.org/10.1007/s10994-010-5176-9, https://doi.org/10.1007/s10994-010-5176-9
3. Banks, J.S.: Sophisticated voting outcomes and agenda control. Social Choice and Welfare **1**(4), 295–306 (1985)
4. Brandt, F., Brill, M., Harrenstein, P.: Tournament solutions. In: Brandt, F., Conitzer, V., Endriss, U., Lang, J., Procaccia, A.D. (eds.) Handbook of Computational Social Choice, pp. 57–84. Cambridge University Press (2016). https://doi.org/10.1017/CBO9781107446984.004, https://doi.org/10.1017/CBO9781107446984.004
5. Charbit, P., Thomassé, S., Yeo, A.: The minimum feedback arc set problem is np-hard for tournaments. Combinatorics, Probability & Computing **16**, 1–4 (2007). https://doi.org/10.1017/S0963548306007887, https://doi.org/10.1017/S0963548306007887
6. Copeland, A.H.: A reasonable social welfare function. Tech. rep., mimeo, 1951. University of Michigan (1951)
7. Gutin, G.Z., Mertzios, G.B., Reidl, F.: Searching for maximum out-degree vertices in tournaments. CoRR **abs/1801.04702** (2018), http://arxiv.org/abs/1801.04702
8. Herlocker, J.L., Konstan, J.A., Terveen, L.G., Riedl, J.: Evaluating collaborative filtering recommender systems. ACM Trans. Inf. Syst. **22**(1), 5–53 (2004). https://doi.org/10.1145/963770.963772, https://doi.org/10.1145/963770.963772
9. Hudry, O.: A survey on the complexity of tournament solutions. Mathematical Social Sciences **57**(3), 292–303 (2009). https://doi.org/10.1016/j.mathsocsci.2008.12.002, https://doi.org/10.1016/j.mathsocsci.2008.12.002
10. Laslier, J.F.: Tournament solutions and majority voting. No. 7, Springer Verlag (1997)
11. Maarek, Y.: Alexa and her shopping journey. In: Proceedings of the 27th ACM International Conference on Information and Knowledge Management. pp. 1–1. CIKM '18, ACM, New York, NY, USA (2018). https://doi.org/10.1145/3269206.3272923, http://doi.acm.org/10.1145/3269206.3272923
12. Moon, J.W.: Topics on tournaments. Holt, Rinehart, and Winston (1968)
13. Reid, K.B.: Tournaments. In: Gross, J.L., Yellen, J., Zhang, P. (eds.) Handbook of Graph Theory, 2nd edition. Chapman and Hall/CRC (2013)
14. Shen, J., Sheng, L., Wu, J.: Searching for sorted sequences of kings in tournaments. SIAM J. Comput. **32**, 1201–1209 (2003). https://doi.org/10.1137/S0097539702410053, https://doi.org/10.1137/S0097539702410053
15. Slater, P.: Inconsistencies in a schedule of paired comparisons. Biometrika **48**(3/4), 303–312 (1961)