

# The Traveling Tournament Problem: Rows-First versus Columns-First

Kristian Verduin<sup>1,2</sup>, Ruben Horn<sup>2,3</sup>, Okke van Eck<sup>1,2</sup>,  
Reitze Jansen<sup>1</sup>, Thomas Weise<sup>4</sup> and Daan van den Berg<sup>1,2</sup>

<sup>1</sup>Department of Computer Science, University of Amsterdam, The Netherlands

<sup>2</sup>Department of Computer Science, VU Amsterdam, The Netherlands

<sup>3</sup>Helmut-Schmidt-University, Hamburg, Germany

<sup>4</sup>Institute of Applied Optimization, Hefei University, China

**Keywords:** Traveling Tournament Problem, Genetic Algorithms, Evolutionary Computing, Constraints, Constraint Hierarchy.

**Abstract:** At the time of writing, there is no known deterministic time algorithm to uniformly sample initial valid solutions for the traveling tournament problem, severely impeding any evolutionary approach that would need a random initial population. Repeatedly random sampling initial solutions until we find a valid one is apparently the best we can do, but even this rather crude method still requires exponential time. It does make a difference however, if one chooses to generate initial schedules column-by-column or row-by-row.


## 1 INTRODUCTION


Even though both are proven to be NP-hard (Thielen and Westphal, 2011; Verduin et al., 2023a), the traveling tournament problem (TTP) is much harder than the likesounding traveling salesman problem (TSP). Being NP-hard means any exact algorithm requires exponential time<sup>1</sup>, but for the TSP, we can at least uniformly sample random solutions in deterministic linear time, and perform mutations that transitively connect all valid solutions in deterministic constant time. The availability of an efficient sampling algorithm and efficient transitive mutations for TSP make the problem amenable to a broad variety of evolutionary algorithms. For TTP, both such algorithms are currently not available, and it is really the question whether they ever will.


The root of the issue lies in the constraints. The TTP entails scheduling a tournament of an even num-


ber of (baseball) teams ( $n_{teams}$ ), containing  $2 \cdot n_{teams} - 2$  rounds (Easton et al., 2001). Each team needs to play all other teams exactly *twice* in the schedule (once at home, once away), which is known as the **double round-robin** constraint. Additionally, when team A plays team B in one round, the inverse match (B playing A) cannot be scheduled in the immediate consecutive round, which is known as the **noRepeat** constraint. Finally, there is the maximum number of consecutive games any team can play at home (or away), the **maxStreak** constraint<sup>2</sup>. Usually,  $maxStreak = 3$  meaning any team can at most play three consecutive rounds at home or away anywhere in the schedule (Thielen and Westphal, 2011). Only three constraints, but they can be violated many times per schedule (Fig.1, especially for larger numbers of  $n_{teams}$ . And as constraints go, it only takes one violation to render the entire schedule invalid.


But the actual problem is not about satisfying these constraints. The TTP, like the TSP, has a distance matrix which holds the travel time between stadiums, and the main task is actually to minimize the total travel time. The three constraints however, are so asphyxiating, that travel time optimization almost becomes auxiliary to finding a valid schedule in the first


<sup>a</sup> <https://orcid.org/0009-0005-8754-7635>

<sup>b</sup> <https://orcid.org/0000-0001-6643-5582>

<sup>c</sup> <https://orcid.org/0000-0002-3600-5183>

<sup>d</sup> <https://orcid.org/0009-0007-0029-2882>

<sup>e</sup> <https://orcid.org/0000-0002-9687-8509>

<sup>f</sup> <https://orcid.org/0000-0001-5060-3342>

<sup>1</sup>In the worst case instance, and assuming  $P \neq NP$

<sup>2</sup>Terminology varies slightly across literature.



Figure 1: **Left:** A 4-team TTP schedule, generated in rows-first order. It has one *doubleRoundRobin* violation in column 1 (playing the Giants away twice), one *maxStreak* violation in column 4, and no less than 4 *noRepeat* violations on row 5 in all columns. **Right:** Internal representation of the same schedule. The padding the first column of the matrix with unused zeroes assures that all team numbers are nonzero, and thereby facilitates the use of negative numbers in the source code.

place. This is a real head breaker because NP-hard problems, practically unsolvable by exact algorithms, are usually attacked with some kind of metaheuristic, which usually requires an initial population of valid individuals, preferably uniformly sampled from the solution space, to even start its optimizing process.

## 2 SAMPLING TTP SOLUTIONS

To be completely clear: we think that uniform random sampling, which is so much needed for unbiased operation of metaheuristic algorithms, is not possible in deterministic polynomial time for TTP. Furthermore, we suspect that ‘eligible’ mutations, that connect all valid solutions into one traversable neighbourhood, might not be possible either. Finally, we think that an ‘eligible’ crossover operator might also not exist. We are aware that this is a very strong and potentially falsifiable position, but we also think that the current state of consensus pertaining the TTP needs it. The challenge to our colleagues therefore is: prove us wrong, and bring your best game. Supply either a deterministic polynomial sampling algorithm, an eli-

gible mutation or crossover operator. To the best of our knowledge, none such methods exist.

Of course we dove into literature, and we encountered a lot of recurring patterns. Many of the studies use small values for  $n_{teams}$ , often from Michael Trick’s benchmark (Trick, 2022). Many of these even smaller than the real-life Major League baseball, which ‘only’ holds 30 teams at the time of writing. The world cup soccer however, typically played in multiple countries, will hold 48 teams from 2026 onward, and is expected to grow from there.

While multiple of the papers implicitly or even explicitly acknowledge the sparseness of the solution landscape, none of them report the number of infeasible schedules that are generated. This suggests two things:

1. New schedules (e.g. by mutation or crossover) are made in *stochastic* time: if a mutation produces an invalid schedule, simply retry. This is different, and less reliable, than a mutation in deterministic time, such as a 2-opt in the traveling salesman problem.
2. The number of necessary tries in obtaining a valid mutated schedule might increase exponentially,

similar to the generation of random initial valid solutions as reported by Verduin et al. (Verduin et al., 2023a). A very disheartening outlook might therefore be that for larger problem instances, even simply mutating a valid schedule into another valid schedule in deterministic feasible time is not possible. In some cases, we think this might have been observed but not reported by the authors, as it would explain the low number of  $n_{teams}$  in many studies.

Often, a non-uniform initialization procedure is used (usually the polygon method (de Werra, 1988; Dinitz et al., 2006)), which is a deterministic scheduling process, but of course only produces solutions from a tiny portion of the combinatorial space. It is noteworthy however that the original proposal of this method by (de Werra, 1988) considers creating a schedule with one game between any two opponents and allows breaks.

Finally, a number of papers pertain the *mirrored* traveling tournament problem (mTTP), in which the lower half of the schedule is identical to the upper half, with the home/away designations reverted. Although this subset of TTP-solutions has a (super)exponentially smaller search space than the regular TTP, it is still estimated to be far beyond the reaches of explicit enumeration for any reasonable values of  $n_{teams}$ . But it might still be easier, even for metaheuristic algorithms, because the *doubleRoundRobin* constraint is less likely to be violated. An open question however, is to what extent *that* observation ties into the results reported in our study.

### 3 RELATED WORK: EAs FOR TTP

In a 2006 study, Biajoli and Lorena apply a genetic algorithm together with simulated annealing to the mTTP in which the *doubleRoundRobin* constraint is satisfied by repeating the first half of the schedule with home/away designations reversed (Biajoli and Lorena, 2006). The population is initialized using the polygon method (de Werra, 1988) by expansion from a compact chromosome representation which is a permutation of the teams. Their possible mutations are: swapping the home/away roles for a match, swapping two matches, or swapping the entire match plan for the two teams. The last mutation can produce invalid schedules, causing a cascade of changes to the schedule which is not further elaborated upon in this paper. During the initialization, only the home/away

swap mutation is applied using a randomized non-ascend method. Recombination is implemented using block order crossover, whereby the parent from which to inherit a gene is determined by a uniform random bitmask, resulting in an increase from  $O(n)$  to  $O(2^n)$  potential different offspring compared to one-point crossover (Syswerda, 1989). An individual produced by recombination may undergo the mutations of swapping home and away roles for a match, swapping opponents for a match, or the entire itinerary for two teams. This *can* result in invalid individuals, which is not further addressed in the paper (Ribeiro and Urrutia, 2007). Before selection, the local search is applied using simulated annealing, and the home/away and team swap mutations. Both feasible and infeasible schedules are considered as the ‘neighborhood’ of an individual in this step. Their largest instance comes from real world sports and has 24 teams, which, according to the authors, is large compared to other instances used in the literature. We agree.

In another study from 2006, Anagnostopoulos et al. designate *doubleRoundRobin* as a hard constraint, while *noRepeat* and *maxStreak* are considered soft constraints, and then apply simulated annealing with reheating to the TTP (Anagnostopoulos et al., 2006). Soft constraints may be violated as the algorithm traverses the search space, and both feasible and infeasible neighborhood of schedules are explored. Using three mutation operations being the swapping two rounds, swapping the home/away roles in one pair of games, swapping opponents for a pair of teams, or swapping rounds for one or two teams. The number of violations is incorporated into the objective function with a parameter  $\omega$  to balance the exploration of both feasible and infeasible regions. The initial schedule is generated in a greedy recursive algorithm, and their maximum  $n_{teams}$  is 16, taken from Trick’s benchmark. To us, this sounds like a promising approach, but the big question is how long the algorithm stays in the invalid space for values of  $\omega$ , and whether a suitable value for the parameter actually exists.

Tajbakhsh et al. present an approach with particle swarm optimization improved by simulated annealing to the TTP modelled using binary integer programming in which infeasible schedules are penalized (Tajbakhsh et al., 2009). The generated individuals are only guaranteed to satisfy one hard constraint, being the *doubleRoundRobin* constraint. The neighborhood explored by simulated annealing is generated using the same three mutation operations as in Anagnostopoulos et al. (Anagnostopoulos et al., 2006). The largest instance from (Trick, 2022) used for evaluation has 10 teams. We suspect that the TTP is sig-

nificantly easier without its *maxStreak* constraint.

Uthus et al. propose ant colony optimization with forward checking and conflict-directed backjumping for the TTP (Uthus et al., 2009). Schedules are generated round by round from the team with the fewest possible opponents remaining. Unsafe backjumping, at the cost of potentially missing feasible schedules, helps the algorithm to leave unfeasible partial ones. The algorithm generates and applies all possible home/away sequences for teams not violating the corresponding constraint. Feasible solutions are improved using tabu search. This approach is evaluated using instances of up to 20 teams from Trick's benchmark.

Nitin Choubey proposes symbolic chromosomes with a unique character for each team as an encoding scheme of all matches in a single string for TTP using a genetic algorithm (Choubey, 2010). The random initialization process is not described in detail and may consist of forming a random permutation of all pairs of teams. The approach uses bit-swap mutation and two-point crossover with internal swapping. Violations of constraints are penalized in the fitness calculation. The inclusion of the required number of slots in the penalty suggests that their method also creates schedules with 'holes' such that not every team plays a match in every round. It is evaluated using TTP instances of up to just 8 teams from Trick's online benchmark repository (Trick, 2022).

Saul and Adewumi investigate an artificial bee colony algorithm for the TTP (Saul and Adewumi, 2012). They use the polygon method to create initial schedules, and their experiment uses a population of 20 individuals over 20 cycles (only). Similar to Anagnostopoulos et al., they use mutation operations for swapping home/away configurations, opponents, rounds and a partial swap of rounds but not of teams. Their largest test instance consists of 16 teams, but they note that "the algorithm [does] not perform well on larger instances". To mitigate this, they propose to develop more the neighborhood relations to search through, however, they do not mention their approach to infeasible schedules.

Gupta et al. present a grey wolf optimizer for the TTP (Gupta et al., 2015). The polygon method is used to initialize the population, and simulated annealing using the same five mutation types as Saul and Adewumi (Saul and Adewumi, 2012) to optimize the schedules. For the TTP, individuals are updated by changing a single position in the initial permutation of teams according to the best schedule. If a feasible schedule cannot be generated from this, the move is not applied. Unfortunately, they do not report how often this happens, a common theme in TTP papers.

Besides, for larger number of  $n_{teams}$ , the algorithm might be stuck in invalid space forever, and maybe therefore the approach is evaluated using instances from Trick's benchmark with a size of only up to 16 teams. In their conclusion, the authors acknowledge that the *doubleRoundRobin* constraint presents a significant difficulty and hint at the possible unfeasibility of generating schedules in real world scenarios within a short timeframe. We agree with this observation; it could well be that without the *doubleRoundRobin* constraint, finding valid TTP-schedules is easy.

Rutjanisarakul and Jiarasuksakun also tackle the mTTP using a genetic algorithm (Rutjanisarakul and Jiarasuksakun, 2017). They separate the binary home/away and categorical opponents for all matches into two matrices, but the algorithm only operates on the former. During initialization, the second half of the teams in the home/away matrix are assigned the inverse of the first half. Mutation is applied by flipping the binary value in the same cell of all four quadrants. Recombination is performed by selecting the first half of the teams using crossover and again filling the second half with the inverse, as in the initialization. The opponent matrix is then created using an iterative scheduling algorithm. They evaluate their approach using TTP instances up to  $n_{teams} = 20$ . The authors acknowledge that crossover without filling the second half of the teams with the inverse of the first may result in invalid schedules, requiring repeated attempts. However, they do not address the fact that the mutation operation could also introduce violations and how they handle this.

Khelifa et al. propose an "enhanced genetic algorithm for the TTP" (Khelifa et al., 2017). They initialize the population using the polygon method (de Werra, 1988; Dinitz et al., 2006) and then choose the best schedule from its neighborhood created by mutation. They use the same mutations as Baijoli et al. (Biajoli and Lorena, 2006), but introduce a new crossover that takes the partial itinerary with the lowest cost from the first parent and iteratively generate the remaining rounds from a minimum weight pairing of the graph of teams, with the away/home assignment from the second parent if possible. Additionally, their method includes variable neighborhood search consisting of the mutations described previously. Their test instances are taken from Trick's benchmark and have at most 10 teams. They note that "it is not easy to create the remaining rounds of an incomplete schedule without breaking the [*doubleRoundRobin*] constraint", but the treatment of invalid schedules is not described further. Also, the mutation types as described on page 2 of their paper might introduce violations, but we were

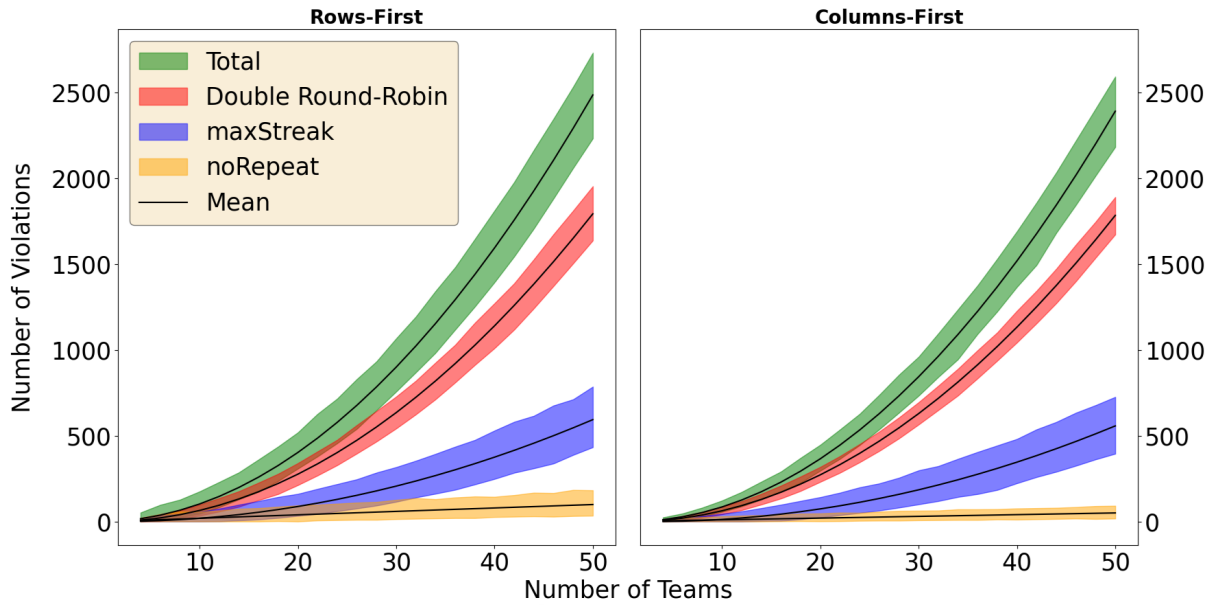


Figure 2: Whether random TTP schedules are generated in rows-first or columns-first order, the number of total violations increases quadratically. However, the number of *noRepeat*-violations is almost half for columns first, but the effect on its linear increase is barely noticeable against the quadratic total.

as yet unable to reach the authors for comments or source code.

A recent approach by Halder et al. compares a genetic algorithm and a particle swarm optimization for the TTP (Halder et al., 2022). Algorithms use the same representation as Choubey for their candidate schedules, but, since it is not described how new generations are created or how violations are treated, we assume that it is also similar to the approach taken in that paper. Supposedly, the particle swarm outperforms the genetic algorithm on the test instances with up to  $n_{teams} = 16$  from teams Trick’s benchmark.

#### 4 EXPERIMENT: ROWS-FIRST VERSUS COLUMNS-FIRST

So as it turns out, we have a problem on our hands: the TTP cannot be solved by exact algorithms for any feasible  $n_{teams}$  because it is NP-hard, but it can ALSO not be solved by evolutionary algorithms, since they need a uniformly sampled random initial population of solutions – which can also not be done in any sort of feasible time. At the time of writing, it seems that uniform random sampling initial solutions to the TTP can only be done in stochastic time: resampling time and time again, hoping to find a solution with zero constraint violations. Sadly, this stochastic time appears to be of exponential nature (Verduin et al., 2023b; Verduin et al., 2023a), making it practically

impossible to solve TTP-instances with any substantial  $n_{teams}$ .

Nonetheless, we would like to press efforts in this direction, and the most straightforward approach is to build up a schedule round by round. A single round can be randomly generated in deterministic linear time: first make a ‘remaining’-list of all teams, then randomly select a team, remove it from the list<sup>3</sup>, randomly select a second team, also remove that team from the ‘remaining’-list and insert these as opponents in the round, one of the two randomly assigned their home venue, the other as playing away. Repeat the procedure by choosing a second pair of opponents, then a third pair and so on until the list is empty. This round-by-round, or “rows-first” generation guarantees, in deterministic linear time, that every round in itself is valid: every team plays exactly one opponent, which is not itself, and home and away assignments are conflict-free. As each round is uniform randomly generated, the resulting schedule is also fully uniformly random, but nonetheless not free of violations.

An alternative approach would be to do a column-by-column (“columns-first”) generation. Each column should hold all teams twice (once at home, once away), except for the column’s own team, which can not play against itself. A columns-first generation can also be done in deterministic linear time, which in-

<sup>3</sup>This intermittent removal step might look superfluous, but is necessary for the deterministic-time claim.

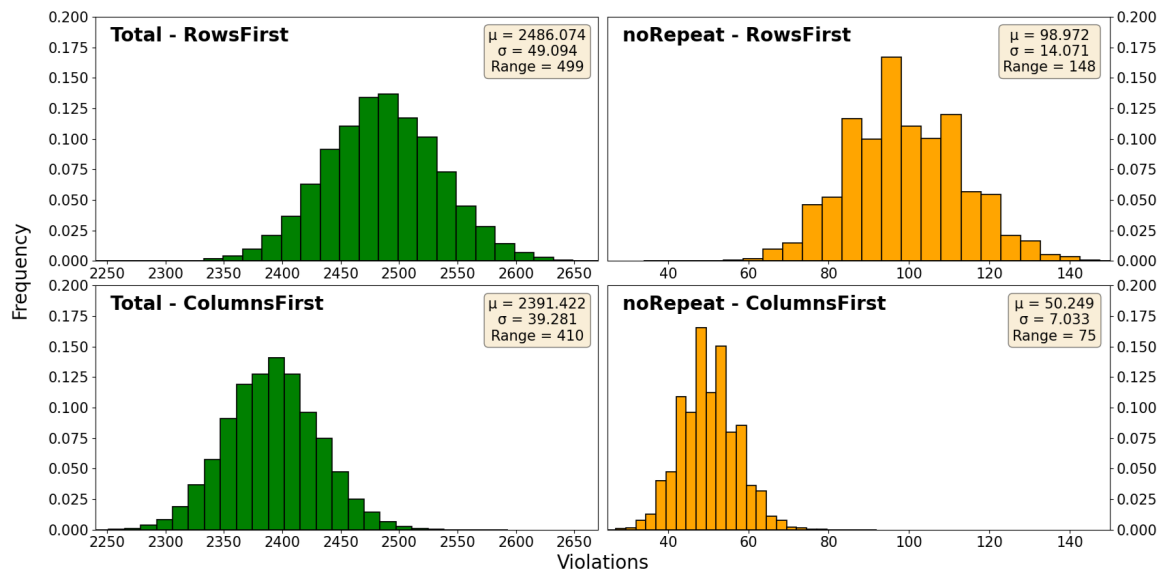


Figure 3: Using rows-first generation or columns-first generation for TTP schedules has a significant impact on the number of *noRepeat* violations for the traveling tournament problem. The number of total violations however, remains largely the same, even though the range of values decreases with approximately 20%. In this figure,  $n_{teams} = 50$ , and both experiments consist of 5 million samples.

volves creating a ‘remaining’-list for team  $T$ . Other than in rows-first generation, it is specific to teams  $T$  by containing all teams twice, once at home and once away but not  $T$  itself. Internally, the ‘remaining’-list is a set containing numbers  $\{1, 2, \dots, n_{teams}\}$  twice: positive for an ‘at home’-opponents, negative for an ‘away’-opponent. For this reason, the list contains no zeroes, and the zero-column of the matrix containing the schedule is padded with unused zeroes (Fig. 1). It also allows the integer value in a matrix cell to be immediately used as a column number for opposing teams. A for-loop randomly picking a team from the ‘remaining’-list, placing them in the schedule in the first open slot of column  $T$ , and then removing them from the same list, effectively generating a random permutation in deterministic linear time. Of course, this can be done in stochastic time too, with a while-loop, but we find that approach sloppy, and possibly more time consuming. After the procedure is completed for the last column, the resulting complete schedule is uniformly random, in deterministic linear time, but again not free of violations. Strangely enough though, the number and type of violations created by columns-first differs substantially from rows-first.

Our experiment entails generating 240 million random TTP schedules: 5 million for each  $n_{teams} \in \{4, 6, 8, \dots, 48, 50\}$  using the rows-first method totalling to 120 million, and another 120 million for the same  $n_{teams}$ , each 5 million, but with columns-first. After completely filling up a schedule, the constraint viola-

tions are counted. All work was done in 48 threads on SURF’s Snellius Compute Cluster<sup>4</sup> running Debian Linux in approximately 30 hours, and our Python source code is publicly available (Anonymous, 2024).

For the *noRepeat* constraint, a violation is counted for every team if a round’s opponent is the same as the opponent from the previous round, regardless of the home or away designation: playing the same team in two subsequent rounds means one more *noRepeat* violation. Internally, this is done by a for-loop that compares the absolute values (thereby neglecting the home/away designations) of two consecutive rows in a column. For the *maxStreak = 3* constraint, every home or away game after the third is counted as one constraint violation.

The *doubleRoundRobin* constraint is a little less straightforward though. Historically, constructive approaches (such as Frohner et al.’s beam search (Frohner et al., 2020)) use round-first generation only hold valid partial schedules, i.e. having zero violations. But when it comes to allowing invalid schedules, counting *doubleRoundRobin* violations is not entirely straightforward. Departing from the only known previous counting methods (Verduin et al., 2023b) and (Verduin et al., 2023a), we count the *doubleRoundRobin* constraint violations as follows: every column  $T$  should hold every team twice, once home once away, (internally denoted as once posi-

<sup>4</sup><https://www.surf.nl/en/dutch-national-supercomputer-snellius>

tive, once negative), except  $T$  itself. For every team *missing*, one column-violation is counted. Simultaneously, every row should hold each team exactly once, in either home or away assignment (internally represented by the absolute value). For each team missing from a round, one row violation is counted. The number of *doubleRoundRobin* violations is then the sum of all row violations and all column violations. It should be noted however, that columns-first produces only row violations, and the rows-first produces only column violations. It is an open problem whether a deterministic time algorithm exists that does produce a uniform random schedule with neither column violations nor row violations. We think it does not, and also have doubts on whether it even *can* exist.

## 5 RESULTS

After the 5 million rows-first random schedules for a value of  $n_{teams}$  were generated and violation-counted, the mean number of violations was taken for each violation type (*doubleRoundRobin*, *maxStreak*, *noRepeat*, *total*) for every  $n_{teams}$ , and characterizations were made through polynomial function fitting (Fig. 2). The same technique was then applied for columns-first generation, yielding another set of polynomial characterizations.

The results show a very clear pattern: for the expected number of *doubleRoundRobin* violations  $O(0.74(n_{teams})^2)$  and *maxStreak* violations  $O(0.25(n_{teams})^2)$ , it really doesn't matter whether to use rows-first or columns-first generation – their characterizations are almost identical. Although columns-first generation is slightly better than rows-first overall, the difference converges to 1% for *doubleRoundRobin* and  $< 6\%$  for *maxStreak*. But remarkably enough, this does not hold for the number of *noRepeat* violations which is almost 94% *higher* when using rows-first generation, than with columns-first generation, in  $O(7.88 \cdot n_{teams})$  and  $O(4.07 \cdot n_{teams})$  respectively. For all eight characterizations, the  $R^2$  quality-of-fit was at least 0.9999.

Another striking difference is in the range (max-min) and the standard deviation ( $\sigma$ ) for the total number of violations (Figure 3). Although both are significantly higher in rows-first, both values appear to converge to 125% of the values for columns-first, even though this percentage is naturally a bit wobblier for the range, which depends on individual outliers.

## 6 INTERMEZZO: TO MATCH OR NOT TO MATCH

During the finalization of this manuscript, one discussion popping up between the authors was the asymmetry of *matching* between rows-first and columns-first. In rows-first, the teams are selected as pairs, inserted in both corresponding columns, one randomly designated as ‘away’, the other ‘at home’. This means that for rows-first, not only the appearance of “every team once per row” is satisfied, but also their opposition in pairs, and even their home-away designation.

These possible violations were not counted, and thereby might present columns-first as more favourable than it actually is. On the other hand, it might be possible to do the matching for columns-first *also*, although at the time of writing we are not sure whether it leads to better schedules, or to a deadlock later in the assignment.

This incompleteness of thought was one reason not to endeavour in this direction yet, but another reason is given by the existence of (approximate) random sampling of magic squares (Jacobson and Matthews, 1996). Though the approach is not without difficulties, it is principally possible to uniformly randomly sample magic squares. Although it is possible to transform any *mirrored* TTP schedule into a magic square, the converse is not necessarily true: not every magic square is transformable into a (mirrored) TTP-schedule – even though some are. Still, we feel that these results are so deeply connected they could shed new light on the problem of uniformly randomly sampling TTP schedules. The current way of counting violations in TTP is almost directly suitable for counting violations in randomly sampled magic squares. We will explore these, and several other avenues, in future work.

## 7 CONCLUSION

Columns-first generation is better than rows-first generation. It scores lower for all violation types, but whereas the difference in *doubleRoundRobin* violations and *maxStreak* violations tends to zero for increasing  $n_{teams}$ , the true gain for columns-first is in *noRepeat*, which converges to about half the number compared to rows-first – for the same time complexity.

Great, we just cut the amount of expected *noRepeat* violations in half. But does it really strike a dent in the problem as a whole? Both the number of *doubleRoundRobin* violations and *maxStreak* violations increase quadratically, dwarfing the linearly

Table 1: Expected numbers of violations can be characterized by a quadratic polynomial ( $x$  substitutes  $n_{teams}$  for readability). The resulting functions are nearly identical for either generation method, except for the expected number of *noRepeat* violations, which doubles from columns-first to rows-first. Note that the exact same function was fit to all three violation types, but for *noRepeat*, the first constant just fit to zero, dropping the quadratic term.

	Rows-First	Columns-First
<i>Total</i>	$0.9855x^2 + 0.4725x - 1.3570$	$0.9862x^2 - 1.5050x + 1.1980$
<i>dRR</i>	$0.7357x^2 - 0.9178x - 0.0610$	$0.7360x^2 + 1.1140x - 0.1436$
<i>maxStreak</i>	$0.2500x^2 - 0.6251x - 0.0021$	$0.2502x^2 - 1.3890x - 1.3180$
<i>noRepeat</i>	$2.0150x - 1.2840$	$0.9855x + 0.0239$

increasing number of *noRepeat* violations as  $n_{teams}$  increases. So for specific obscure variants of the TTP it might help a bit, but for now, the progress on just generating valid initial solutions for the TTP is very meagre, practically speaking. Still, it is a principal step forward in finding a feasible-time random sampling algorithm for the problem.

A more important takeaway from this investigation is that it is very hard, if not impossible, to generate uniform random valid TTP-schedules in any sort of deterministic time, even when completely ignoring the *maxStreak* and *noRepeat* constraints. For the best uniform algorithm we know, columns-first generation, the expected number of constraint violations for the TTP still increases quadratically in the number of teams. In computer science, a polynomial increase is usually considered innocuous but remember that in this case, *all* constraints need to be satisfied before even considering a minimized travel distance – the real task at hand. Besides, the stochastic quadratic increase in violations might give rise to a stochastic *exponential* number of required samples for a finding a single valid initial schedule, as suggested by earlier studies (Verduin et al., 2023b; Verduin et al., 2023a). Furthermore, the TTP is not the only problem having this difficulty; HP-protein folding also seems to be affected by it, but to a lesser extent (Dill, 1985; van Eck and van den Berg, 2023; Jansen et al., 2023).

It is for this reason, the unavailability of a feasible random sampling algorithm (in either stochastic or deterministic time), that we argue that the TTP is harder than the TSP, even though both are listed as NP-hard. The TSP can still be ‘solved’ with hill climbing, simulated annealing or evolutionary algorithms (Koppenhol et al., 2022), whereas the TTP probably cannot, simply because random initial solutions can not be produced in feasible time. And whether eligible feasible-time mutations and crossovers exist also remains to be seen. It looks like finding valid schedules *in itself* is quite a challenge, and might be amenable to approaches such as frequency fitness assignment or local optima network analysis before even thinking about minimizing travel

distance (de Bruin et al., 2023; Liang et al., 2022; Thomson et al., 2023; Thomson et al., 2022).

## ACKNOWLEDGEMENT

Logos in this paper were remade by melling2293@Flickr, and distributed under creative commons license. We also thank Reviewer#1 from ICEIS 2024 for reading our paper so well.

## REFERENCES

- Anagnostopoulos, A., Michel, L., Hentenryck, P. V., and Vergados, Y. (2006). A simulated annealing approach to the traveling tournament problem. *Journal of Scheduling*, 9(2):177–193.
- Anonymous (2024). Repository containing source material: <https://anonymous.4open.science/r/TTP-Column-vs-Row-Constraints-46FE/README.md>.
- Biajoli, F. L. and Lorena, L. A. N. (2006). Mirrored traveling tournament problem: An evolutionary approach. In Sichman, J. S., Coelho, H., and Rezende, S. O., editors, *Advances in Artificial Intelligence - IBERAMIA-SBIA 2006*, pages 208–217, Berlin, Heidelberg. Springer Berlin Heidelberg.
- Choubey, N. (2010). A novel encoding scheme for traveling tournament problem using genetic algorithm. *International Journal of Computer Applications*, ecot.
- de Bruin, E., Thomson, S. L., and Berg, D. v. d. (2023). Frequency fitness assignment on jssp: A critical review. In *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*, pages 351–363. Springer.
- de Werra, D. (1988). Some models of graphs for scheduling sports competitions. *Discrete Applied Mathematics*, 21(1):47–65.
- Dill, K. A. (1985). Theory for the folding and stability of globular proteins. *Biochemistry*, 24(6):1501–1509.
- Dinitz, J. H., Froncek, D., Lamken, E. R., and Wallis, W. D. (2006). Scheduling a tournament. In *Handbook of Combinatorial Designs*. Chapman and Hall/CRC.
- Easton, K., Nemhauser, G., and Trick, M. (2001). The traveling tournament problem description and benchmarks. In *Principles and Practice of Constraint Pro-*



- gramming—CP 2001: 7th International Conference, CP 2001 Paphos, Cyprus, November 26–December 1, 2001 Proceedings 7, pages 580–584. Springer.
- Frohner, N., Neumann, B., and Raidl, G. (2020). A beam search approach to the traveling tournament problem. In *Evolutionary Computation in Combinatorial Optimization*, pages 67–82. Springer International Publishing.
- Gupta, D., Anand, C., and Dewan, T. (2015). Enhanced heuristic approach for traveling tournament problem based on grey wolf optimizer. In *2015 Eighth International Conference on Contemporary Computing (IC3)*, pages 235–240.
- Haldar, A., Mondal, S., Mukherjee, A., and Chatterjee, K. (2022). A comparative analysis of application of genetic algorithm and particle swarm optimization in solving traveling tournament problem (ttp). *International Journal of Bioinformatics and Intelligent Computing*.
- Jacobson, M. T. and Matthews, P. (1996). Generating uniformly distributed random latin squares. *Journal of Combinatorial Designs*, 4(6):405–437.
- Jansen, R., Horn, R., van Eck, O., Verduin, K., Thomson, S., and van den Berg, D. (2023). Can hp-protein folding be solved with genetic algorithms? maybe not. In *Proceedings of the 15th International Joint Conference on Computational Intelligence ECTA - Volume 1*, pages 131–140.
- Khelifa, M., Boughaci, D., and Aïmeur, E. (2017). An enhanced genetic algorithm with a new crossover operator for the traveling tournament problem. In *2017 4th International Conference on Control, Decision and Information Technologies (CoDIT)*, pages 1072–1077.
- Koppenhol, L., Brouwer, N., Dijkzeul, D., Pijning, I., Slegers, J., and Van Den Berg, D. (2022). Exactly characterizable parameter settings in a crossoverless evolutionary algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1640–1649.
- Liang, T., Wu, Z., Lässig, J., van den Berg, D., and Weise, T. (2022). Solving the traveling salesperson problem using frequency fitness assignment. In *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 360–367. IEEE.
- Ribeiro, C. C. and Urrutia, S. (2007). Heuristics for the mirrored traveling tournament problem. *European Journal of Operational Research*, 179(3):775–787.
- Rutjanisarakul, T. and Jiarasuksakun, T. (2017). A sport tournament scheduling by genetic algorithm with swapping method. *Journal of Engineering and Applied Sciences*, 13.
- Saul, S. and Adewumi, A. (2012). An artificial bees colony algorithm for the traveling tournament problem. In *41st Annual Conference of the Operations Research Society of South Africa*, page 10.
- Syswerda, G. (1989). Uniform crossover in genetic algorithms. In *Proc. 3rd Intl Conference on Genetic Algorithms 1989*.
- Tajbakhsh, A., Eshghi, K., and Shamsi, A. (2009). A hybrid pso-sa algorithm for the traveling tournament problem. In *2009 International Conference on Computers & Industrial Engineering*, pages 512–518.
- Thielen, C. and Westphal, S. (2011). Complexity of the traveling tournament problem. *Theoretical Computer Science*, 412(4-5):345–351.
- Thomson, S. L., Ochoa, G., and Verel, S. (2022). The fractal geometry of fitness landscapes at the local optima level. *Natural Computing*, pages 1–17.
- Thomson, S. L., Veerapen, N., Ochoa, G., and van den Berg, D. (2023). Randomness in local optima network sampling. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, pages 2099–2107.
- Trick, M. A. (2022). Challenge traveling tournament problems.
- Uthus, D. C., Riddle, P. J., and Guesgen, H. W. (2009). An ant colony optimization approach to the traveling tournament problem. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, GECCO '09*, page 81–88, New York, NY, USA. Association for Computing Machinery.
- van Eck, O. and van den Berg, D. (2023). Quantifying instance hardness of protein folding within the hp-model. (accepted for publication at CIBCB'23).
- Verduin, K., Thomson, S. L., and van den Berg, D. (2023a). Too constrained for genetic algorithms. too hard for evolutionary computing. the traveling tournament problem. In *Proceedings of the 15th International Joint Conference on Computational Intelligence ECTA - Volume 1*, pages 246–257.
- Verduin, K., Weise, T., and van den Berg, D. (2023b). Why is the traveling tournament problem not solved with genetic algorithms?