Optimizing Positioning in Soccer

Tiago Mendes-Neves
LIAAD-INESC TEC
Faculty of Engineering, University of Porto
Porto, Portugal
tiago.m.neves@inesctec.pt

***Abstract*—Sports are requiring an ever-increasing level of opponent analysis to prepare a game. Some of these tasks focus on the same principles: finding the correct positioning to counter the opposing team offense. In this paper, we propose a methodology to obtain a solution for the defensive positioning of a team on an opponent goal kick. We built a framework that allows us to define a game situation for which we want to find solutions. Then, we test and compare the performance of several optimization methods. We conclude that using Hill Climbing is the best approach to use, with some niche use cases depending on criteriums such as a limited number of iterations or requiring the best solution possible.**

***Index Terms*—soccer, set-pieces, artificial intelligence, optimization**

I. Introduction

Artificial intelligence is entrenching itself in sports at a rapid pace. One field that has seen little research in sports is defensive positioning. Meanwhile, the preparation of a game of soccer has increased in complexity in the last 20 years. As an example, Manchester United has more than doubled its employees in technical and coaching roles [1]. Many of these employees perform tasks where artificial intelligence techniques can increase the quickness and quality of work. We propose using optimization algorithms to search for defensive strategies in soccer to assist technical staff in game preparation.

The use of computational techniques to obtain solutions regarding player positioning has been addressed extensively in competitions like RoboCup [2]. The proposed solutions use optimization algorithms, such as Particle Swarm Optimization and Genetic Algorithms. The approaches try to either maximize the number of pass lines blocked [3], blocking the view of a player to a point of interest, minimizing the distance to the opposition, or simply following the rules of the game [4]. More advanced works use Reinforcement Learning techniques that optimize goal difference, regions occupied, and ball possession [5], [6], reducing the distance between a target object and a goal, and pass direction [7]. Similar to our use case, some works focus on set pieces [3], [8]. Optimization algorithms have many other use cases such as trajectory planning [9].

We extend the current proposed optimization targets by adding pitch control [10], a metric that allows us to quantify what is the area of the pitch dominated by a team. Team A dominates an area if the player closer to this area belongs to team A. There are several extensions to this work that take into account the speed and acceleration of the players. We did not use these extensions since they would increase substantially the complexity of the work we want to perform.

Currently, there is no consensus regarding the best approach for solving the problem we propose. Furthermore, few solutions address the 10 vs. 10 complexity problem but use the somewhat easier 5 vs. 5 scenarios.

In this work, we test several optimization algorithms intending to find which approach leads to the best results. For this, we provide a framework that allows the user to set the positions of the offensive team against whom the algorithms will try to optimize. We conclude that local search methods are the better option to find solutions to our problem, with Hill Climbing approaches edging out other approaches in nearly every point of view.

Summary of the contributions:

- We describe the algorithms used in Section II.
- We define the framework, fitness functions, and parameters used in Section III.
- We present the results obtained and discuss the performance of the algorithms in detail in Section IV.
- We present our final remarks in Section V.

II. Methods

This section briefly explains the algorithms used in this work.

A. Random Search

Random Search, described in Algorithm 1, is a direct search approach that consists of randomly generating solutions to our problem. This method has no guarantee of convergence or finding a good solution unless we perform it infinitely.

Algorithm 1 Random Search Pseudocode

Input: Stopping criteria
Output: Best solution found

```
while Stopping criteria not met do
    Generate random solution
    if Solution is better than current best then
        Store solution as current best
    end if
end while
```

This work is financed by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within project UIDB/50014/2020.

We will use the Random Search method to establish a baseline to compare with the remaining algorithms. If an algorithm cannot improve over the Random Search baseline over any time frame, the algorithm cannot find a reasonable solution to our problem.

Ideally, we would also check the Grid Search solution to establish a topline. Grid Search checks every possible solution, which guarantees an optimal solution. However, the amount of possible solutions to our problem is 10e40. The time requirement would be on the scale of 10e39 seconds, which is infeasible.

B. CMA-ES

CMA-ES (Covariance Matrix Adaptation Evolution Strategy) [11] is an evolutionary algorithm that samples a new candidate solution according to a multivariate normal distribution, defined by a covariance matrix. The covariance matrix is updated after each iteration to create a more accurate distribution, increasing the probability of finding good solutions.

In this work, we use the CMA-ES version present in the Optuna python library [12].

C. TPE

TPE (Tree-structured Parzen Estimators) [13] algorithm is an iterative process that, similarly to CMA-ES, creates a distribution that suggests the following solution to evaluate.

To calculate which point to query next, TPE splits the data from previous experiments into two distributions: the best results, $g(x)$, and the worst results $l(x)$. To find the best point to query, we find the value x that minimizes their coefficient $g(x)/l(x)$.

In this work, we use the TPE version present in the Optuna python library [12].

D. Hill Climbing

Hill Climbing, described in Algorithm 2, is a local search algorithm that attempts to find a better solution by making incremental changes to the current solution and following the best solution.

Algorithm 2 Hill Climbing Pseudocode

Input: Step, Stopping criteria (optional)
Output: Best solution found

```
while Solution improved OR Step is not at minimum do
    Generate all successors
    Calculate fitness for all successors
    Find best successor
    if Best successor is better that current best then
        Store best successor as current best
    else
        Decrease step
    end if
end while
```

This algorithm faces some problems: (1) It converges to local minimums, (2) it can get stuck in plateaux, and (3) it might not be able to find the solution due to ridges [14].

To address the problems described, we opted to use a version of Hill Climbing with decreasing step. This version starts by searching for solutions that far away from the current solution. As the number of iterations increases or the algorithm reaches a plateau, the step decreases. This method avoids (1) converging to the nearest local minimum and (2) getting stuck in a plateau since the algorithm can make large steps away from the current solution, only converging to a local solution after finding a good starting point. The method can also find a solution in a ridge due to the step decreasing to a low value.

E. Simulated Annealing

Simulated Annealing, described in Algorithm 3, leverages the Hill Climbing concept of local search and introduces an ingredient that allows the algorithm to escape local minimums [14].

Algorithm 3 Simulated Annealing Pseudocode

Input: Step, Temperature, Temperature decay, Stopping criteria (optional)
Output: Best solution found

```
while Solution improved OR Step is not at minimum do
    Generate all successors
    Calculate fitness for all successors
    Find best successor
    if Best successor is better that current best then
        Store best successor as current best
    else
        Decrease step
    end if
    Adopt random solution with probability given in function of the difference in fitness and temperature
end while
```

This ingredient is a random move generated according to a probability defined by the temperature t and accepted with probability $p(t, \Delta q)$, where Δq is the difference between the fitness of the current solution and the randomly generated solution. The more a random solution improves the current solution, the more likely to accept it as the current solution. As the temperature decreases over time, the randomly generated solution will be less likely to be accepted, allowing the algorithm to converge similarly to the Hill Climbing algorithm.

F. Genetic Algorithms

Genetic Algorithm, described in Algorithm 4, is an evolutionary algorithm inspired by the theory of natural evolution of Darwin. This algorithm cycles through generations and changes the population by selecting, crossing, and mutating the individuals. As the generations go by, the individuals' fitness increases, leading to a better solution.

The selection phase is done by randomly eliminating elements of the population that are not considered elitist.

The crossover phase is heavily dependent on the use case, but the primary rationale is to combine two elements of the population to generate a new individual. In the mutation phase,

Algorithm 4 Genetic Algorithm Pseudocode

Input: Stopping criteria (usually Generations to run), Population size, Elitist rate, Survival rate, Mutation rate
Output: Best solution found

```
Generate population of random individuals
for all Generations to run do
    Evaluate all individuals
    Select the fittest individuals and remove individuals from the population according to their fitness
    Generate new individuals by combining selected individuals
    Apply mutations to individuals
end for
```

we change individuals to introduce new random elements in the population. Both the crossover and mutation phases can be combined so that only the new individuals in the population suffer mutations.

III. Methodology

A. The problem

The research question we are answering is the following: given the positions of an attacking team (e.g., Figure 1), what is the best method to define where should the defensive players be positioned?

To answer this question, we will use optimization algorithms to define the position of the 10 defending players. The algorithms will return solutions composed of 10 (x, y) integer coordinates that will constitute a defensive team which we will evaluate through our fitness function. We are excluding the goalkeeper for simplification purposes.

To calculate a neighbor solution, we move a player in one direction according to the defined step (-/+) across both x and y axes. This definition means that every solution has 40 neighbor solutions (10 players * 2 axes * 2 directions).

When generating random solutions, we limit the random solution to contain the all the players in the bounding box of the opposing team, plus a small margin. The random solution is always compliant with the rules (i.e., no defending players inside the goalkeeper box.

B. Fitness Function

The algorithms defined in Section II all rely on being able to obtain feedback that quantifies how good a proposed solution is, know as fitness. In our work, the fitness function has five components:

- **Percentage of covered pass lines:** A pass line (Figure 2) is a straight line between the player that possesses the ball and another player of the same team. This component aims to increase the number of pass lines covered, reducing the offensive team's. We consider a pass line blocked when an opposing player has a distance lower than the Minimum Adversary Distance (MAD). Throughout this work, we use MAD=5.

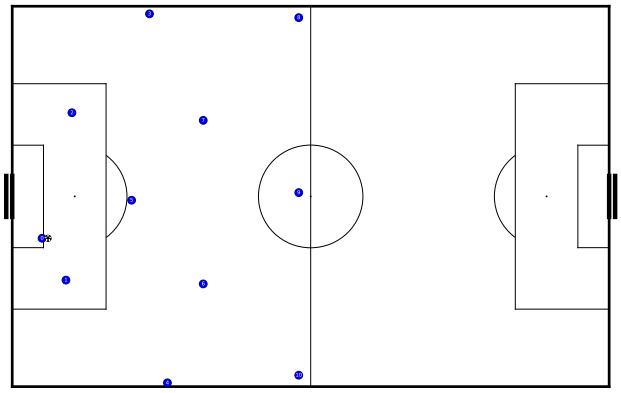

Fig. 1. The position of the attacking team used in the experiments. We emulated one of the most common positions of attacking players in a goal kick.

- **Percentage of covered goal lines:** A goal line (Figure 3) is a straight line between any player of the team with possession and the opposition's goal. This component aims to guarantee that if an offensive player receives the ball, he will be opposed at least once before shooting. Similar to pass lines, a goal line blocked when an opposing player has a distance lower than the MAD.
- **Marking distance:** The marking distance (Figure 4) is the sum of the distances to the closest opposing player for every player of the team with possession. It aims to control the amount of pressure made on the opposing team. The distance between two players is at least 1.
- **Pitch control:** Pitch control (Figure 5) measures the amount of the pitch controlled by a certain team. The definition of control is: if player A' from team A is the closest player to a certain point in the pitch, then this point is controlled by team A. By calculating this for multiple points across the pitch, we can approximate the percentage controlled by each team. The goal is to maximize the area controlled by the defending team. In this work, we give twice the weight to the area between the attacking team's most advanced player (e.g., player 9 in Figure 5) and the defending team's goal.
- **Validation:** This component aims to introduce some nuances that force the solution to be compliant with soccer rules. For example, no players are allowed inside the box in a goal kick situation. Penalizations are cumulative.

To calculate the fitness of our solution, we perform a weighted sum of the five components described above. The user defines the weight given to each component. In this work, all weights are equal to 1, with the exception of pitch control which has weight=4. This is due to pitch control changing much slower than the other components.

C. Parameters

If we do not define a parameter in this section, consider the default value from the library. We ran all algorithms 20

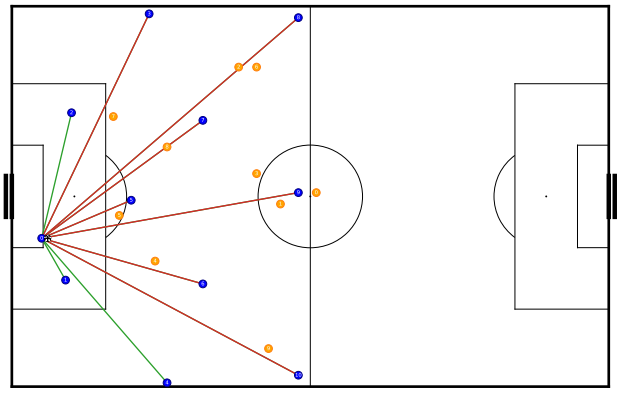

Fig. 2. Example of **pass** lines. Red lines are blocked by the opponent and green lines are clear. This example also shows the result of optimizing a randomized team using the **Hill Climbing** algorithm, using only the **percentage of covered pass lines** as the target for our optimization.

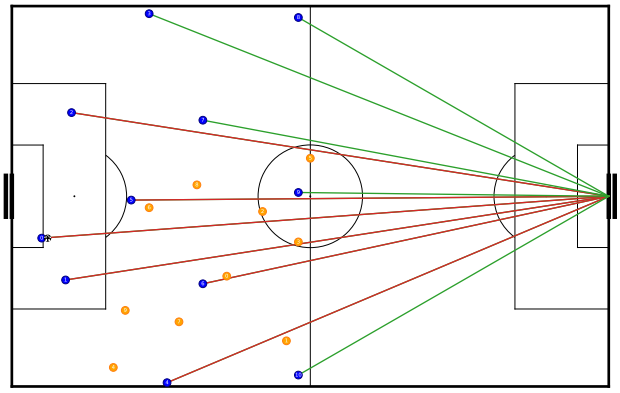

Fig. 3. Example of **goal** lines. Red lines are blocked by the opponent and green lines are clear. This example also shows the result of optimizing a randomized team using the **Hill Climbing** algorithm, using only the **percentage of covered goal lines** as the target for our optimization.

times, except on the generalization tests, where we only ran the algorithms once.

1) Random Search: We used 20 000 iterations.

2) CMA-ES: used 5 000 iterations.

3) TPE: used 5 000 iterations.

4) Hill Climbing: has a initial step of 10 that halves every 125 cycles or when the result cannot be improved. The procedure ends if we reach a step lower than one.

5) Simulated Annealing: uses the same step parameters as the Hill Climbing. The procedure starts at a temperature of 1, decaying at 4% per cycle. until it reaches the minimum temperature of 0.01. The probability of acceptance p is calculated by Equations 2 and 1.

$$\Delta d = fitness(random) - fitness(current\ solution) \quad (1)$$

$$p(acceptance) = t * \frac{arctan(\Delta d * 100) + \pi/2}{\pi} \quad (2)$$

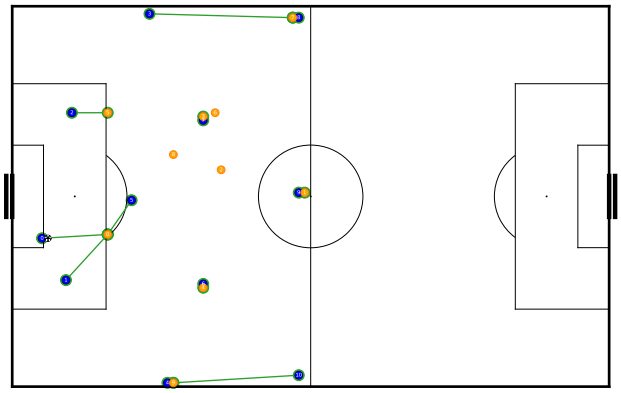

Fig. 4. Example of the calculation method for the **marking distance**. We consider the marking distance the sum of the size of all green lines in the figure. This example also shows the result of optimizing a randomized team using the **Hill Climbing** algorithm, using only the **marking distance** as the target for our optimization.

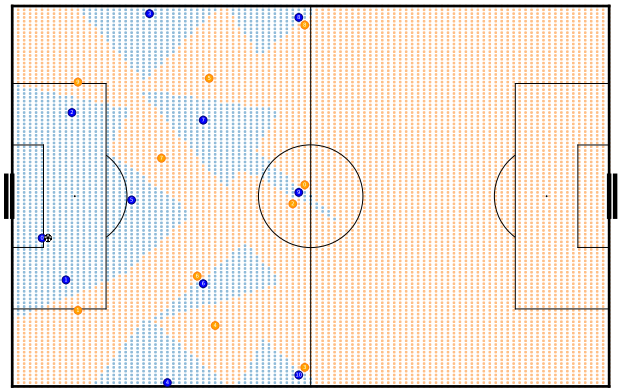

Fig. 5. Example of the **pitch control** metric. We calculate pitch control by summing all the areas covered by the defensive team. The area between player 9 and blue team's is worth twice. This example also shows the result of optimizing a randomized team using the **Hill Climbing** algorithm, using only the **pitch control** as the target for our optimization.

6) Genetic Algorithm: uses a population size of 100. The elitist rate is 5%, the default survival rate is 70%, and the default mutation rate is 20%. The survival rate decreases with the number of iterations without improving (defined as η) (Equation 3), leading to more a aggressive shuffling of the population when it stabilizes. In parallel, the mutation rate also increases with η (Equation 4). The algorithm stops if there is no improvement in 20 generations.

$$survival\ rate = \frac{default\ survival\ rate}{e^{\eta/20}} \quad (3)$$

$$mutation\ rate = default\ mutation\ rate * e^{\eta/20} \quad (4)$$

IV. Results

In this section, we present the results obtained from different perspectives: (1) we present the algorithms' results individu-

ally, (2) we compare the algorithms, and (3) we visualize the solutions presented by the best algorithms.

A. Random Search

Figure 6 presents the results obtained from performing Random Search. Overall, the Random Search method converges pretty quickly to a solution but then struggles to improve from there.

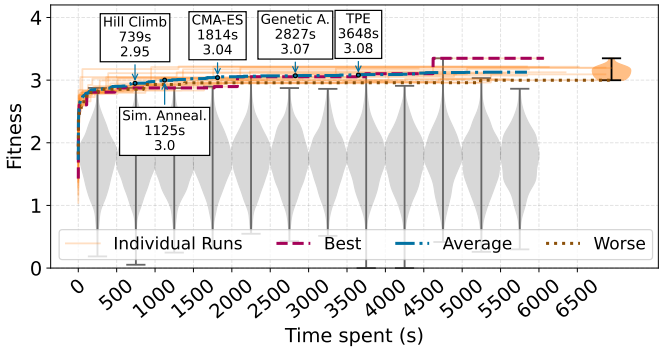

Fig. 6. **Random Search results.** The annotations are the average Random Search's results in the same time it took for the other algorithms to run. The grey violin plots represent the distribution of the fitness of the solutions tested by the algorithm. The orange violin plot is the distribution of the algorithm's final results.

In the grey violin plots, we can observe that this method generates solutions that have fitnesses that are close to normal distributions. This means that, for each instance, we can calculate the probability of the current solution being improved. The experiments resulted in a mean of 1.73 and a standard deviation of 0.41. Table I presents the probabilities of obtaining a result above a certain threshold.

TABLE I
PROBABILITY OF OBTAINING A SOLUTION ABOVE X.

X	p(x>X)
1.73	0.97
2.0	0.78
2.5	0.17
3.0	0.01
3.5	8.73e-05[a]
4.0	2.15e-07[b]

Mean=1.73, Standard Deviation=0.41.
[a] 1 in 11 450 solutions.
[b] 1 in 4 650 000 solutions.

As we can evaluate from Table I, if we can consider solutions that obtain consistent fitness values above 3.5 unlikely to have resulted from chance.

B. CMA-ES

As seen in Figure 7 the CMA-ES algorithm achieves a good balance between following a local solution and searching all the state space. This balance is evident by the number of iterations that it uses early in the procedure to explore a wide range of solutions. This is visible by analyzing the large range of the first violin plot in Figure 7. However, it quickly moves away from these iterations and narrows its focus to more localized solutions.

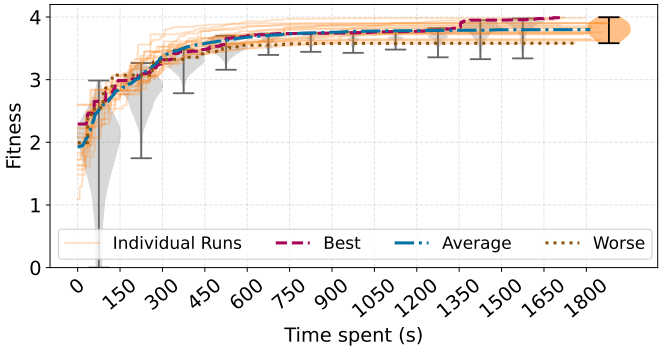

Fig. 7. **CMA-ES results.** We can observe that CMA-ES performs local search as the grey violin plot distributions are skewed to the top.

It is also relevant to refer that after approximately 4000 iterations, CMA-ES has reached 98% of its maximum performance. This threshold is consistent if we verify the experiments in Figure 7. If we stop the algorithm at 4 000 iterations, we can substantially reduce the algorithm's time requirements.

Figure 7 also shows that the CMA-ES consistently obtains good performances. The orange violin plot on the right presents a very small range where the solutions may vary.

C. TPE

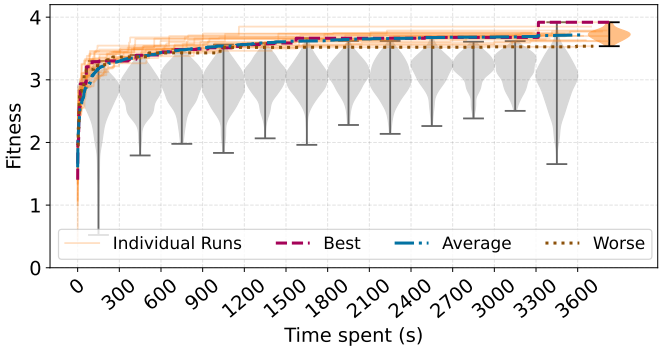

Fig. 8. **TPE results.** TPE does not converge the same way the CMA-ES does. Instead of a high skew towards the top in the grey violin plots, it only has a slight skew.

In opposition to CMA-ES, the TPE algorithm fails to recognize that some of the solutions are not valid and continues to explore them throughout time. In Figure 8, we can observe that TPE spends too many iterations exploring a wide range of solutions. Meanwhile, there is a minimal exploration to improve the local solution, as evidenced by the low skew of the solutions tested towards the top. Although it performs better than Random Search, this method lacks local optimization, which hurts its performance.

In Figure 8, we can observe that TPE is consistent, having a low standard deviation of the results. However, the bar is lowered compared with CMA-ES.

D. Hill Climbing

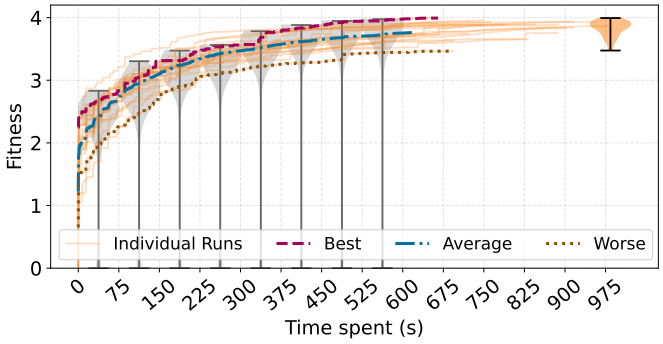

Fig. 9. **Hill Climbing results.** Hill Climbing performs local search for a better solution. The results differ from CMA-ES since the distribution of solutions tested are substantially different. While CMA-ES varies the grey violin plots' distributions **form** quite substantially, the **form** of the Hill Climbing distribution is consistent over time.

The Hill Climbing algorithm performed as expected: first, it improves quickly due to a larger step, and then it converges to a local solution as the step decreases. We can observe this in Figure 9.

In opposition to CMA-ES and TPE, the Hill Climbing algorithm does not require a set number of iterations at which it will stop. In Figure 9, we can see that the number of iterations required by this algorithm varies substantially. From the quickest to the slowest run, the algorithm nearly doubles the number of iterations required. Although we did not do it, we can set a predefined number of iterations for the algorithm to stop, increasing its temporal consistency at the cost of a worse solution.

E. Simulated Annealing

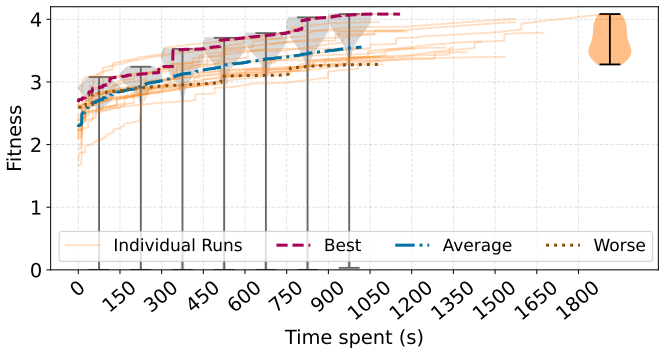

Fig. 10. **Simulated Annealing results.** The grey violin plots follow the same patterns of Hill Climbing.

Analogously to the Hill Climbing results, the Simulated Annealing also focuses on exploring the neighborhood with occasional jumps to solutions depending on the current temperature. In general, these jumps allow Simulated Annealing to outperform Hill Climbing. By observing Figure 10 we can see the similarities between both Simulated Annealing and Hill Climbing.

In Figure 10 we can observe that the Simulated Annealing algorithm used approximately double the iterations used by the Hill Climbing algorithm. The fact that we need to wait for the temperature to lower means that this algorithm might not be suitable for applications that rely on real-time decision-making.

F. Genetic Algorithm

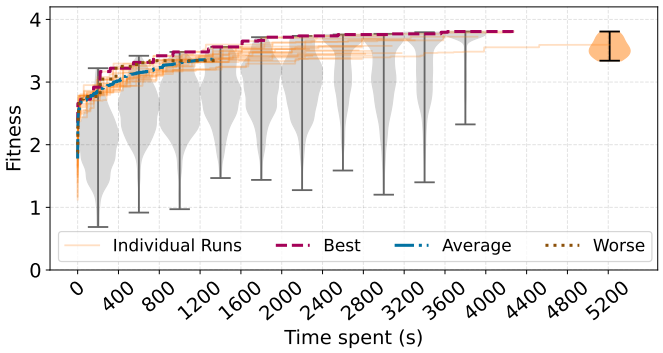

Fig. 11. **Genetic Algorithm results.** The grey violin plots' distribution is very dynamic throughout time.

The evolution of the Genetic Algorithm is more chaotic than the previous local methods explored, as seen in Figure 11. This chaotic nature of the algorithm puts its performance close to the performance of TPE. It also maintains the wastage of iterations throughout the whole procedure.

In Figure 11, we can observe that the results obtained by the Genetic Algorithm are consistent in terms of the standard deviation of the result. However, the number of iterations needed is quite large. In fact, of the algorithms used, this is the algorithm that requires the most iterations.

G. Comparing algorithms

Figure 12, 13, 14 and 15 present the results from the best iterations of each algorithm.

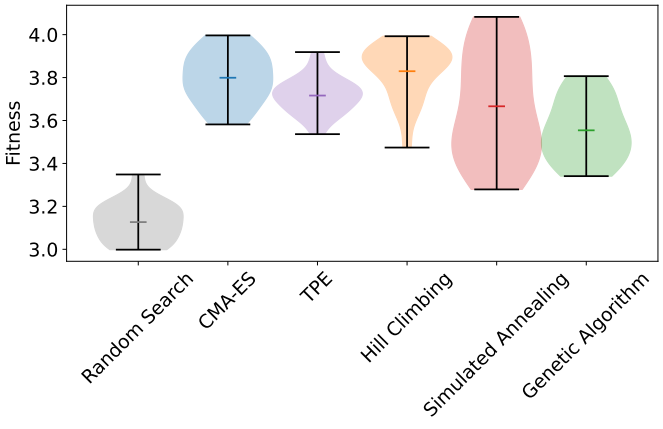

Fig. 12. The distribution of the results for each algorithm. Large colored line represents the mean value.

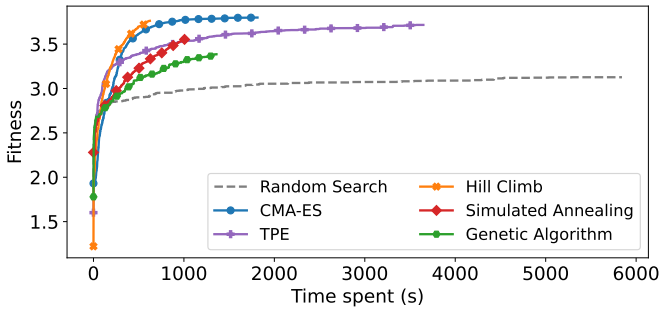

Fig. 13. Average fitness of each algorithm, over time.

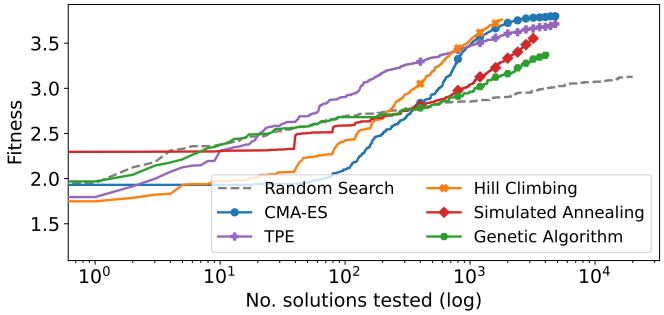

Fig. 14. Average fitness of each algorithm, over iterations. The X axis is in logarithmic scale.

As we can see, the best algorithm depends on the point of view:

- Regarding pure performance, the Simulated Annealing is the winner, achieving higher fitness than any other algorithm. (Figure 12)
- If we analyze the curvatures in Figure 13, we observe that (1) Hill Climbing achieves the best average run and (2) does it in the shortest time.
- Regarding quickness (<20 iterations), no algorithm can significantly distinguish itself from Random Search. (Figure 14)
- The interval between 20 and 800 iterations is won by TPE, which contradicts some of our earlier findings. We believe this to be the reason for the algorithms' success on hyperparameter optimization tasks in machine learning [12]. (Figure 14)
- From 800 solutions tested onwards, Hill Climbing wins. (Figure 14)
- Regarding the chances of obtaining bad solutions, Both Simulated Annealing and Genetic Algorithms are riskier, with the lowest possible values of fitness obtained. (Figure 12)
- Random Search is, by far, the worst method in terms of time performance, as expected. However, TPE is the algorithm with the largest overhead (i.e., time spent by the algorithms without considering the time spent evaluating the solutions). In this regard, Hill Climbing is also a winner in this category. (Figure 15)

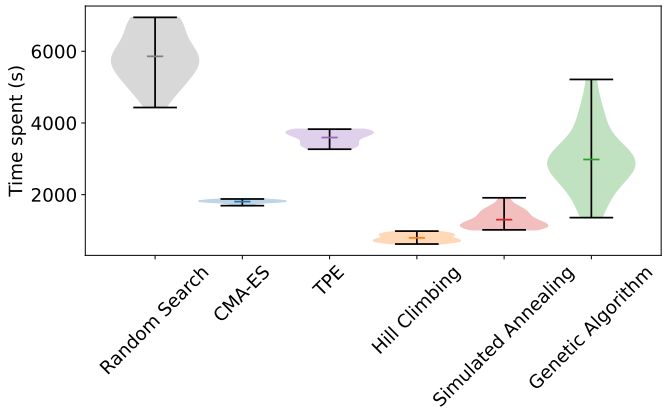

Fig. 15. The distribution of the time spent by each algorithm to find a solution. Large colored line represents the mean value.

Hill Climbing surpassed, by far, the remaining algorithms tested in this work. Not only it performed better on average in terms of fitness, but it was also the quickest to reach a solution.

It makes sense to use search for a solution locally since, in a real scenario, we would start from a predefined solution (e.g., a team's starting formation). Furthermore, if we wish to use this system in real-time, we would only require small changes to the current best solution. Since we start from a good solution, local search methods are the best solution for nearly every use case.

One use case where Hill Climbing would not be the best solution is if we want to find the absolute best solution possible, with little time restrictions. For this, using Simulated Annealing would lead to better results, as observed by the higher ceiling in Figure 12.

Other than these niche use cases, where TPE for small iteration number and Simulated Annealing for global optimum search, Hill Climbing is the recommended solution.

H. Solutions found

In this section we will present two best solutions: the best solution found by Hill Climb (Figure 16) and by Simulated Annealing (Figure 17).

In general, both solutions converged to a solution that passes the eye test. The Hill Climbing proposes an interesting solution where it covers all the players on the side of the ball. The algorithm seems to emphasize reducing the distance rather than optimizing, for example, pitch control.

On the Simulated Annealing solution, the algorithm does not properly cover important areas. It essentially segments the team in two blocks, leaving a lot of space between the lines. Similar to the Hill Climbing solution, it focuses on reducing the distance rather than occupying relevant spaces.

Rather than blaming the algorithms for these faults, we believe that the reason for these suboptimal solutions is the rather simplistic fitness function used to guide the algorithms. The fitness function (1) does not properly quantify how certain

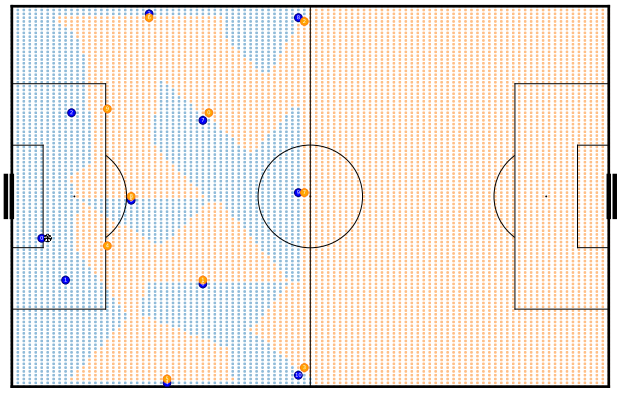

Fig. 16. The best solution found by the Hill Climbing algorithm. Blue circles represent the attacking team, and the orange circles represent the defending team positioned by the algorithm. The light dots represent which team dominates this specific area of the pitch (pitch control).

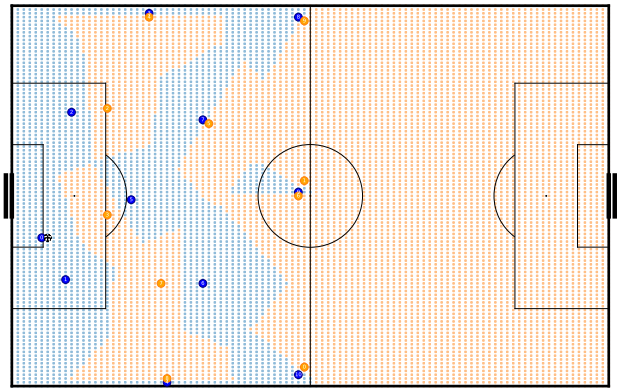

Fig. 17. Best solution found by the Simulated Annealing algorithm.

relevant areas of the pitch are, (2) does not find a proper trade-off between marking distance and pitch control, over-relying on the former, and (3) takes into account the current pass lines but not future ones.

Even with this study's limitations, the solutions found are not far off what usually happens in reality. While the solutions are not perfect and the heuristic fails to evaluate some situations correctly, the solutions evaluated indicate that we are not far off being able to design supportive tools for helping prepare the game plan.

V. Conclusion

In this work, we reviewed the performance of optimization algorithms to solve defensive player positioning in soccer. We concluded that the best solution was to use Hill Climbing or Simulated Annealing. We believe this is due to the difference between local and global maximums not being very relevant. Furthermore, teams use defined formations that we can use as a starting point for our search, removing the need to explore global solutions.

In the future, we hope to improve the fitness function used in this work. While our fitness function improved the current state-of-the-art, we still feel that it is not adequate for every case. Furthermore, we hope to include more information about the game state, such as the speed and acceleration of the players, to find a solution that considers more factors.

VI. Appendix

A. Repository

All the code is available at: https://github.com/nvsclub/MarkingWithAI. In the img/animated directory there are animations showing the learning process of the algorithm.

B. Extra Results

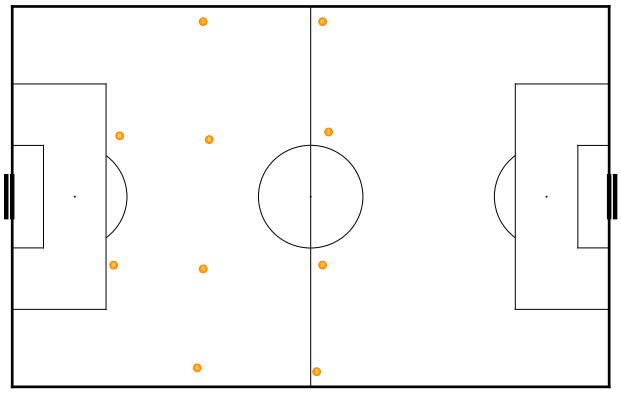

Fig. 18. Starting 4-4-2 formation for the following tests.

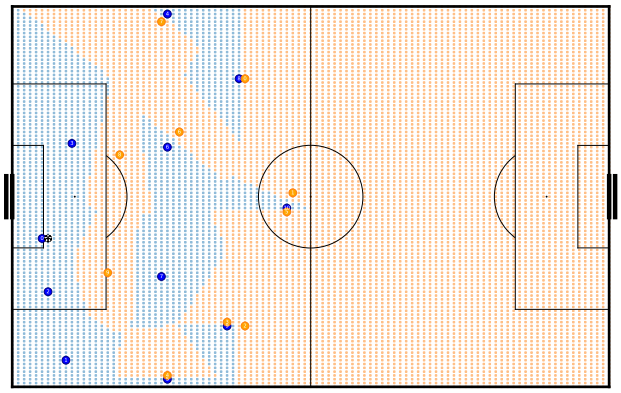

Fig. 19. Solution found by the Simulated Annealing algorithm, starting from a 4-4-2 formation.

References

[1] D. Lange, "Number of employees of manchester united from 2013 to 2020, by segment," Statista, Nov. 26, 2020. [Online]. [Online]. Available: https://www.statista.com/statistics/383940/manchester-united-employees-by-sector/

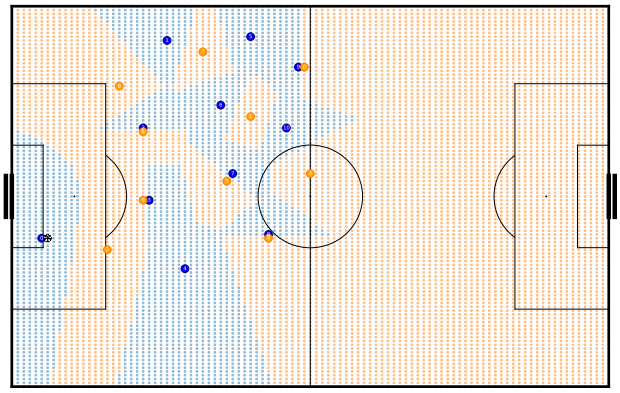

Fig. 20. Solution found by the Simulated Annealing algorithm, starting from a 4-4-2 formation.

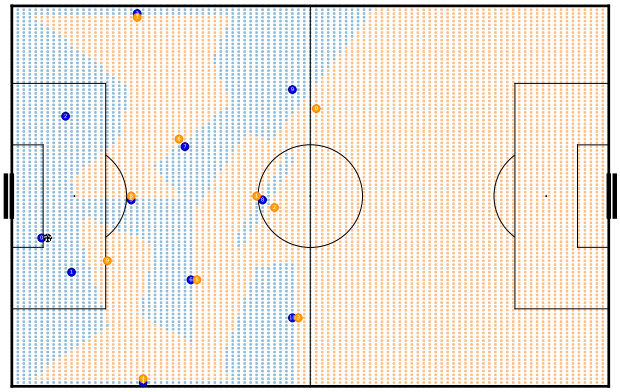

Fig. 21. Solution found by the Simulated Annealing algorithm, starting from a 4-4-2 formation.

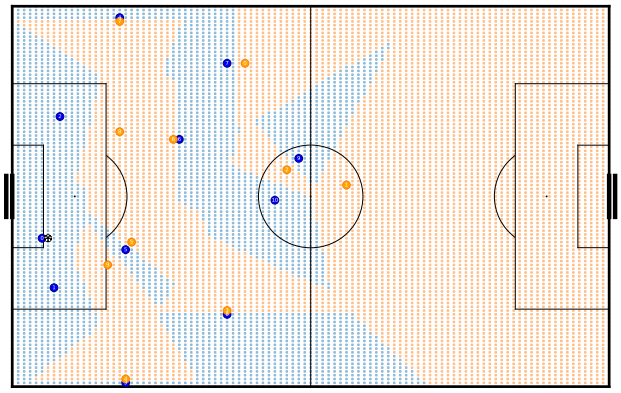

Fig. 22. Solution found by the Simulated Annealing algorithm, starting from a 4-4-2 formation.

[2] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa, "Robocup: The robot world cup initiative," in *Proceedings of the First International Conference on Autonomous Agents*, ser. AGENTS '97. New York, NY, USA: Association for Computing Machinery, 1997, p. 340–347. [Online]. Available: https://doi.org/10.1145/267658.267738

[3] M. A. Pchek Laureano and F. Tonidandel, "Performing and blocking passes in small size league," in *2019 Latin American Robotics Symposium (LARS), 2019 Brazilian Symposium on Robotics (SBR) and 2019 Workshop on Robotics in Education (WRE)*, 2019, pp. 19–24.

[4] M. A. P. Laureano and F. Tonidandel, "Analysis of the pso parameters for a robots positioning system in ssl," in *RoboCup 2019: Robot World Cup XXIII*, S. Chalup, T. Niemueller, J. Suthakorn, and M.-A. Williams, Eds. Cham: Springer International Publishing, 2019, pp. 126–139.

[5] A. S. Larik and S. Haider, "On using evolutionary computation approach for strategy optimization in robot soccer," in *2016 2nd International Conference on Robotics and Artificial Intelligence (ICRAI)*, 2016, pp. 11–16.

[6] A. Larik and S. Haider, "A framework based on evolutionary algorithm for strategy optimization in robot soccer," *Soft Comput.*, vol. 23, no. 16, p. 7287–7302, Aug. 2019. [Online]. Available: https://doi.org/10.1007/s00500-018-3376-6

[7] M. Abreu, L. P. Reis, and H. L. Cardoso, "Learning high-level robotic soccer strategies from scratch through reinforcement learning," in *2019 IEEE International Conference on Autonomous Robot Systems and Competitions, ICARSC 2019, Porto, Portugal, April 24-26, 2019*, L. Almeida, L. P. Reis, and A. P. Moreira, Eds. IEEE, 2019, pp. 1–7. [Online]. Available: https://doi.org/10.1109/ICARSC.2019.8733606

[8] T. Henn, J. Henrio, and T. Nakashima, "Optimizing player's formations for corner-kick situations in robocup soccer 2d simulation," *Artif. Life Robot.*, vol. 22, no. 3, p. 296–300, Sep. 2017. [Online]. Available: https://doi.org/10.1007/s10015-017-0364-3

[9] A. Agarwalla, A. K. Jain, K. V. Manohar, A. T. Saxena, and J. Mukhopadhyay, "Bayesian optimisation with prior reuse for motion planning in robot soccer," in *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, ser. CoDS-COMAD '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 88–97. [Online]. Available: https://doi.org/10.1145/3152494.3152502

[10] W. Spearman, "Physics-based modeling of pass probabilities in soccer," in *MIT SLOAN 2017*, 2017.

[11] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, pp. 159–195, 2001.

[12] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[13] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger, Eds., vol. 24. Curran Associates, Inc., 2011.

[14] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Upper Saddle River, New Jersey: Prentice Hall, 2003.