

# Documentation with Reference to the Paper

Title: *A Sport Tournament Scheduling by Genetic Algorithm with Swapping Method*  
(Pages 5796–5800)

## Initial Population Generation 📄 .1

:Paper Reference 📄

• "Page 5797, 2nd column, paragraph titled "Genetic Algorithm"

A set of random initial solutions is generated by assigning matches to rounds ensuring that each match"  
"...occurs once

:Code Reference 📄

```
def create_individual():  
    ...  
    matches_copy = matches.copy()  
    random.shuffle(matches_copy)
```

:Explanation ✅

You replicate the paper's approach of generating random valid individuals as initial solutions by assigning  
each match uniquely to available time/venue slots with constraints

## Fitness Function and Conflict Resolution 🧠 .2

:Paper Reference 📄

• "Page 5797, under "Fitness Evaluation"

The fitness value is calculated based on the number of violations, such as... multiple matches by a team"  
"...in one round, and matches in the same venue

:Code Reference 📄

```
if len(set(played_days)) < len(played_days):  
    penalty += 50  
...  
if len(matches_at_slot) > 1:  
    penalty += 100 * (len(matches_at_slot) - 1)
```

:Explanation ✅

Just like in the paper, your fitness function penalizes schedule violations including team duplication and  
venue conflicts

### Swapping Method (Mutation) 🔗 .3

:Paper Reference 📄

:Page 5798, 1st column, under "Swapping Method" •

The swap operation is performed by exchanging two matches in a schedule... this helps to reduce"  
".conflicts

:Code Reference 📄

```
def swap_mutation(schedule, mutation_rate=0.1):  
    ...  
    schedule[idx1], schedule[idx2] = schedule[idx2], schedule[idx1]
```

:Explanation ✅

You've implemented the Swapping Method as part of the mutation phase, which the paper suggests as a  
".means of improving solution quality

### Crossover Mechanism 🔗 .4

:Paper Reference 📄

:Page 5797, last paragraph •

In crossover operation, two parent schedules are recombined to produce new children... segments are"  
".swapped while maintaining feasibility

:Code Reference 📄

```
def order_crossover(parent1, parent2, crossover_rate=0.8):  
    ...  
    offspring1 = [match for match in parent2 if match not in segment]
```

:Explanation ✅

You apply **Order Crossover**, a common method that aligns with the general crossover logic discussed in the  
".paper (though the paper doesn't specify OX by name)

## Selection and Elitism 🏆 .5

:Paper Reference 📄

:Page 5797, last paragraph •

"The best solutions are carried over to the next generation to retain the quality of solutions"

:Code Reference 📄

Edit 🗎

Copy 📋

python

```
def tournament_selection(...)  
def survivor_selection(...)
```

:Explanation ✅

Your use of tournament selection and survivor elitism reflects the paper's emphasis on preserving the best  
.solutions across generations