AI-Driven Optimization for Soccer Scheduling

Melissa Hoang

University of Calgary, Computer Science Department

1 Introduction

n this project, we aim to address the intricate problem of scheduling soccer games and practices for various leagues and divisions within the City of Calgary. The challenge involves managing a diverse set of hard and soft constraints to ensure an optimal and feasible schedule. The hard constraints include requirements such as preventing scheduling conflicts between games and practices, adhering to the city's rule that games on Mondays must also occur at the same time on Wednesdays and Fridays (and similarly for Tuesday/Thursday games), and ensuring that only the specified game or practice slots are utilized. Soft constraints, on the other hand, involve preferences such as avoiding certain time slots, pairing specific games and practices together, and fulfilling minimum slot utilization requirements. Each game and practice has specific preferences and undesired time slots, making the problem highly combinatorial.

The objective of this project is to create a scheduling system that can take input in the form of time slot availability, game and practice requirements, and constraints, and output an optimized schedule. The system will consider penalties for violating soft constraints, such as preferences and unfulfilled pairings, while strictly adhering to the hard constraints. Our approach includes using search algorithms to generate possible schedules, evaluating them based on a penalty score, and refining the schedules to achieve the best possible balance between satisfying constraints and minimizing penalties.

We employ a hybrid approach that combines a set-based genetic algorithm with an or-tree-based search model to effectively tackle the complex scheduling problem of soccer games and practices in Calgary. The genetic algorithm is particularly well-suited for optimizing soft constraints, allowing us to explore a vast solution space while efficiently balancing multiple preferences and requirements. To address hard constraints, we incorporate an or-tree model that generates feasible solutions through systematic exploration of possible game and practice assignments. This model leverages a structured tree framework where each node

represents a partial assignment, and branches explore various continuations while adhering to the defined constraints. By integrating these two methodologies, our approach can iteratively refine schedules, ensuring that games and practices are allocated to optimal time slots while minimizing conflicts and adhering to both league rules and participant preferences. This comprehensive strategy allows for a robust and adaptable scheduling solution, making it suitable for the dynamic nature of sports event management.

2 The Model

Model: $A = (S, T)$ The model defines the main data structure and possibilities. Data Structure - Slots:
Given a list of slots defined as follows:

Games on Mon, Wed, Fri	Games on Tue, Thur
$s_1 - s_{12}$	$s_{13} - s_{20}$

Practices on Mon, Wed	Practices on Tue, Thur	Practices on Fri
$s_{21} - s_{32}$	$s_{33} - s_{44}$	$s_{45} - s_{52}$

Game slots on Monday, Wednesday and Friday are represented as a tuple consisting of $s_1 - s_{12}$ such that $s_1 = MWF9:00 - 10:00, s_2 = MWF10:00 - 11:00$, and so on.

Game slots on Tuesday and Thursday are represented as a tuple consisting of $s_{13} - s_{20}$ such that $s_{13} = TR8:00 - 9:30, s_{14} = TR9:30 - 11:00$, and so on.

Practice slots on Monday and Wednesday are represented as a tuple consisting of $s_{21} - s_{32}$ such that $s_{21} = MW9:00 - 10:00, s_{22} = MW10:00 - 11:00$, and so on.

Practice slots on Tuesday and Thursday are represented as a tuple consisting of $s_{33} - s_{44}$ such that $s_{33} = TR9:00 - 10:00, s_{34} = TR10:00 - 11:00$, and so on.

Practice slots on Friday are represented as a tuple consisting of $s_{45} - s_{52}$ such that $s_{45} = F8:00 - 10:00, s_{46} = TR10:00 - 12:00$, and so on.

Data Structure - Games & Practices:
A tuple of games and practices, GP, is defined as follows: $GP = (gp_1, ..., gp_n)$
$= (g_1, ..., g_m, p_{11}, ..., p_{1k}, ...p_{m1}, ..., p_{mk_m})$
where $\forall j \in \mathbb{N}, 1 \leq j \leq m, g_j \in Games$
and $\forall j' \in \mathbb{N}, 1 \leq j' \leq k_m, p_{jk_m} \in Practices$
Facts: $F = \{(MWF_G, TR_G, MW_P, TR_P, F_P)$ where $MWF_G = (s_1^1, s_2^1, ...), TR_G = (s_1^2, s_2^2, ...), MW_P = (s_1^3, s_2^3, ...), TR_P = (s_1^4, s_2^4, ...), F_P = (S_1^5, S_2^5, ...),$
$\forall j \in \mathbb{N}, 1 \leq j \leq |MWF_G|, s_j^1 \in Slots$ and $assign(gp_j) = s_j^1$,
$\forall j \in \mathbb{N}, 1 \leq j \leq |TR_G|, s_j^2 \in Slots$ and $assign(gp_{|MWT_G|+j}) = s_j^2$,
$\forall j \in \mathbb{N}, 1 \leq j \leq |MW_P|, s_j^3 \in Slots$ and $assign(gp_{|MWT_G|+|TR_G|+j}) = s_j^3$,
$\forall j \in \mathbb{N}, 1 \leq j \leq |TR_P|, s_j^4 \in Slots$ and $assign(gp_{|MWT_G|+|TR_G|+|MW_P|+j}) = s_j^4$,

$\forall j \in \mathbb{N}, 1 \leq j \leq |TR_P|, s_j^5 \in Slots$ and $assign(gp_{|MWT_G|+|TR_G|+|MW_P|+|TR_P|+j}) = s_j^5$,

The set of facts it defined as the slots assigned for practices and games per week grouped together as blocks. Given our previous definition of GP, each element s in $MWF_G, TR_G, MW_P, TR_P, F_P$ corresponds to a game or practice $gp \in GP$, meaning that $assign(gp, s)$ and $assigned(MWF_G, TR_G, MW_P, TR_P, F_P)$.

- Set of possible states: $S \subset 2^F$
- Transitions between states: $T = \{(S, S')|\exists A \rightarrow B \in Ext \quad with \quad A \subseteq S \quad and \quad S' = (S - A) \cup B\}$
- Extension Rules: $Ext \subseteq \{A \rightarrow B|A \subseteq F, \quad B \subseteq F, \quad Mutate(A, B) \vee SwapMutate(A, B) \vee Crossover(A, B) \vee Delete(A, B) \vee Random(A, B)\}$
  The definitions of the above rules are defined below in Section 4.

3 Search Process

- Search Process: $P = (A, Env, K)$
- Env : environment of process
- Search Control: $K = S \quad x \quad Env \rightarrow S$
  $K(s, e) = (s - A) \cup B \quad where \quad s \in S \quad and \quad e \in Env,$
  $A \rightarrow B \in Ext$
- $f_{wert} : 2^F \quad x \quad 2^F \quad x \quad Env \rightarrow \mathbb{N}$
  if —s— = 0, Random(A,B)
  if —s— $\geq$ threshold, Delete(A,B)
  else
  50% of the time, Crossover (P, Q, C)
  30% of the time, Mutate (A, B)
  20% of the time, SwapMutate(A, B)
- $f_{select} : 2^F \quad x \quad 2^F \quad x \quad Env \rightarrow 2^F \quad x \quad 2^F$
  - Delete(A,B):
    threshold = $\frac{\Sigma_{\forall f \in F} Penalty(f_n)}{|F|}$
    $C = \{f|f \in F \, and \, Penalty(f) > \; threshold\}$

A = a random 20% of C
Threshold is the average penalty of all the assignments in A. A random 20% of all the individuals with above-average penalty will be selected to be deleted.

- Mutate(A,B)
  $A = \{P\}, B = \{P, C\}$
  $Ranked(L, L_{Ranked})$
  Ranked is defined below
  P = The first element f of L_{Ranked}
  Select the individual with the lowest penalty value for mutation.
- SwapMutate(A,B)
  $A = \{P\}, B = \{P, C\}$
  $Ranked(L, L_{Ranked})$
  P = The first element f of L_{Ranked}
  Select the individual with the lowest penalty value for swap mutation.
- Crossoever(A,B)
  $A = \{P, Q\}, B = \{P, Q, K\}$
  $Ranked(L, L_{Ranked})$
  P = The first element f of L_{Ranked}
  Q = The second element f of L_{Ranked}
  Select the individuals with the lowest penalty values for crossover.
- Random(A,B):
  Apply the or-tree search (defined in section 5) a predetermined amount of times, $N_{pop} \geq 2$ to generate N_{pop} number of random unique solutions.

- Ranked:
  $F \rightarrow (f_1, ..., f_m)$ where $\forall i \in \mathbb{N}, a \leq i \leq m, f_i \in F$
  $Ranked(L, L_{Ranked})$
  $L = (l_1, ..., l_k, ..., l_j, ..., l_m)$ *where* $\forall i \in \mathbb{N}, 1 \leq i \leq m, f_i \in F$
  $L_{Ranked} = (..., l_j, l_k, ...)$ *where* $\forall j, k \in \mathbb{N}, l_j, l_k \in L$ *and* $Pen(l_j) \leq Pen(l_k)$.
  Given a set of facts, the facts are ranked from the lowest penalty to the highest penalty.

- Penalty: $assigned \rightarrow \mathbb{N}$
  Penalty(assigned(SL), pen) where SL = $(MWF_G, TR_G, MW_P, TR_P, F_P)$

  1. $\forall s_i \in Slots$, if (count of s_i in MWF_G, TR_G $<$ $gamemin(s_i)$
     pen = pen + ($gamemin(s_i)$ -count of s_i in MWF_G, TR_G) * $Pen_{gamemin}$
  2. $\forall s_i \in Slots$, if (count of s_i in MW_P, TR_P, F_P $<$ $practicemin(s_i)$
     pen = pen + ($practicemin(s_i)$ -count of s_i in MW_P, TR_P, F_P) * $Pen_{practicemin}$

3. $\forall s \in Slots, \forall preference((gp, s), n)$ where assigned(gp, s) = false, pen = pen + n.
4. $\forall pair(a, b)$ where $a, b \in Games \cup Practice$, if (assign(a) $\neq$ assign(b)) lor ($\neg sametime(assign(a)$,), pen = pen + $pen_{notpaired}$.
5. $\forall g \in Games \wedge$ g is for tier t, $\forall g' \in Games \wedge g'$ is for tier t $land g' \neq g \wedge assign(g') = assign(g)$, pen = pen + $pen_{section}$.

Penalty takes a possible complete assignment of all games and practices, and calculates how desirable this assignment is based on our given soft constraints. We assume that the model allows only positive numbers as penalties. Therefore, the lower the penalty number is, the more "fit" the assignment is.

1. If the number of games assigned to a slot is lower than the slot's gamemin, then we add the number it is lower by, times the predetermined penalty score to the existing penalty number.
2. If the number of practices assigned to a slot is lower than the slot's practicemin, then we add the number it is lower by, times the predetermined penalty score to the existing penalty number.
3. Given a list of preferences of games/practices for slots, for every preference of an assignment that is not met, we add the predetermined penalty score of that preference to the existing penalty number.
4. Given a list of pairs consisting of games/practices that would like to be scheduled to the same slot, for every pair that is not scheduled to the same slot, we add the predetermined penalty score to the existing penalty number.
5. For all games of a tier t, if there exists another game g' that is scheduled to the same slot, but is in a different division and the same tier, for every g' we add the predetermined penalty score.

- assigned: $f \rightarrow \{true, false\}, where\ f \in F$
  assign(A) = true if and only if
  A = $(MWF_G, TR_G, MW_P, TR_P, F_P) = ((s_{l_1}, ..., s_{l_i}), (s_{l_{i+1}}, ..., s_{l_k}), (s_{l_{k+1}}, ..., s_{l_l}), (s_{l_{l+1}}, ..., s_{l_m}))$
  such that $assign(gp_1) = s_{l_1}, ..., assign(gp_m) = s_{l_m}$.
  GP = $(gp_1, ..., gp_m)$ is defined in section 2.
  Given an A consisting of tuples of slots, assigned(A) is true if each slot is assigned to its corresponding game or practice as defined in section 2.

4 Rule Definitions

- Crossover: Crossover(A,B) where
  $A = \{P, Q\}, B = \{P, Q, K\}$, where for some $i, j, m \in \mathbb{N}, 1 \leq i, j, m \leq |P|, |Q|, P = (s_1, ..., s_i, ..., s_j)$ and
  $Q = (s_1, ..., s_i, ..., s_j, ...)$ and
  $gp_m \in Q$ such that gp_m is assigned to s_i and gp_m is assigned to s_j.
  $P \neq Q$
  If gp_m is chosen, then for Q, $assign(gp_m) = s_i$, and for P, $assign(gp_m) = s_j, K = \{(s_1...s_i...s_j), (s_1...s_i...s_j)\}$
  Pass K to the or-tree to check if it meets hard constraints, if not let or-tree modifies K to the next viable solution that's not equal to P or Q.
  Crossover takes two parent individuals, and randomly selects which game/practice to crossover, and places the game/practice from one parent in the slot that the other parent has the game/practice in, and the other parent does the same, placing the game/practice in the corresponding slot to that that contained the practice/game in the other parent.The practice/game chosen will be done so randomly.

- Swap Mutation: SwapMutate(A,B) where
  $A = \{P\}, B = \{P, C\}$
  $P \in F, P = (s_1...s_i...s_j...)$ such that
  $assign(gp_i) = s_j$ and $assign(gp_j) = s_i$ and Constr(C) = true.
  Pass C to the or-tree to check if it meets hard constraints, if not let or-tree modifies C to the next viable solution that's not equal to P.
  SwapMutation takes a parent individual from A and swap the slot between 2 slots or 2 practices. The result of the swap is appended to the list.

- Mutation: Mutate(A, B) where
  $A = \{P\}, B = \{P, C\}$
  $P \in F, P = (s_1...s_i...s_j)$ such that
  $assign(gp_i) = s_i$ and $assign(gp_j) = s_j$ for some $i, j \in \mathbb{N}, 1 \leq i \leq |P|, 1 \leq j \leq |P|, j \neq i$ and
  $((gp_i \in Games) \wedge (gp|j \in Games)) \vee ((gp_i \in Practices) \wedge (gp_j \in Practices))$
  $C = s_1...s_j...s_j...$ such that $assign(gp_i) = s_j, Constr(C) = true.$
  Pass C to the or-tree to check if it meets hard constraints, if not let or-tree modifies C to the next viable solution that's not equal to P.
  Mutation takes a parent individual from A and reassign the slot of a game or practice of the parent by selecting another slot assigned to some other game or practice. The result of the reassignment is appended to the list.

- Deletion: Delete(A,B) where
  $A \subset F, B = \{\}$
  Deletion takes a set of facts and remove them.

- Random: Random(A, B) where
  Apply the Or-tree search a predetermined amount of times, $N_{pop} \geq 2$, to generate N_{pop} number of random unique solutions.
  B = A passed down to the or-tree N_{pop} times.

5 Random: Or-tree Model

5.1 Or-tree-based Search Model

For our Random rule, we would like to use an or-tree model to populate individuals that meet the hard constraints while disregarding the soft constraints.

- Model $A_\vee = (S_\vee, T_\vee$

- Prob: set of problem descirptions
  $pr \in Prob$ where $pr = WMF_G, TR_G, MW_P, TR_P, F_P$ where $MWF_G = (s_1^1, s_2^1, ...), TR_G = (s_1^2, s_2^2, ...), MW_P = (s_1^3, s_2^3, ...), TR_P = (s_1^4, s_2^4, ...), F_P = (S_1^5, S_2^5, ...)$,
  $\forall j \in \mathbb{N}, 1 \leq j \leq |MWF_G|, s_j^1 \in Slots \cup \$$
  $\forall j \in \mathbb{N}, 1 \leq j \leq |TR_G|, s_j^2 \in Slots \cup \$$
  $\forall j \in \mathbb{N}, 1 \leq j \leq |MW_P|, s_j^3 \in Slots \cup \$$
  $\forall j \in \mathbb{N}, 1 \leq j \leq |TR_P|, s_j^4 \in Slots \cup \$$
  $\forall j \in \mathbb{N}, 1 \leq j \leq |TR_P|, s_j^5 \in Slots \cup \$$.
  These are the same definitions for the Facts as in section 2, except that the s1, s2, s3, s4, and s5 can now be \$. \$ indicates that the game / practice is not assigned to any slot yet.

- $Altern \subseteq Prob^+$ :: alternatives relation.
  $Altern(pr, pr_1, ..., pr_n)$ where
  $\forall s \in pr, s_i is the first such that \$ \in s_i$, and
  $\forall s_j \in Slots, \exists pr_j$ that is created by duplicating pr and then replace the first \$ in pr by s_j. s

- $S_\vee \subseteq Otree$: set of possible states, is subset tree structures where
  $(pr, sol) \in Otree$ for $pr \in Prob$, $sol \in \{ues, ?, no\}, b_i \in Otree$

- $T_\vee \subseteq S_\vee \; x \; S_\vee$: transitions between states such that $T_\vee = ¶(s_1, s_2 | s_1, s_2 \in S_\vee$ and $Erw(s_1, s_2)$
- $Erw_\vee$: A relation on Otree
  - $Erw_\vee((pr, ?), (pr, yes)$ if pr is solved
  - $Erw_\vee((pr, ?), (pr, no)$ if pr is unsolvable
  - $Erw_\vee((pr, ?), (pr, (pr_1, ?), ..., (pr_n, ?)))$ if $Altern(pr, pr_1, ..., pr_n)$ holds
  - $Erw_\vee((pr, ?, b_1, ..., b_n), (pr, ?, b_1^{'}, ..., b_n^{'}))$ if for an i, $Erq_\vee(b_i, b_i^{'})$ and $b_j = b_j^{'}$ for $i \neq j$

5.1.1 Or-tree-based Search Process

- Search Process: $P_\vee = (A_\vee, Env, K_\vee$
  Env: environment of process
- Search Control: $K : X \; x \; Env \rightarrow S$
  - $K(s, e) = s^{'} \; where \; (s, s^{'}) \in T, e \in Env$
  - Let $(pr_1, ?), ..., (pr_0, ?)$ be the open leafs in the current state, and $X = (MWF_G, ..., F_P)$ with the same definitions in section 2 such that X is over some discrete value domains $D = \{D_i | i \in \mathbb{N}, 1 \leq i \leq |X|\} \; \forall i \in \mathbb{N}, 1 \leq i \leq |X|, D_i = Slots$.
    $C = \{C_1, ..., C_m\}$ is a set of constraints. Each constraint C_i is a relation of a subset of the vairables, i.e.
    $C_i = R_i(s_{i,1}, ..., s_{i,k})$ where the relation R_i descirbes every value-tupe in $D_{i,x}x...xD_{i,k}$
    Let $const(X_{m_i}) = |\{C_i | C_i \in C, x_1^1, ..., x_n^5 \; fulfills \; C_i\}|$
  - $C_{solved(pr)} = |\{C_i | C_i \in C, x_1^1, ..., x_n^5 \; fulfills \; C_i\}|$
  - Define constraints $C = \{C_1, ..., C_{12}\}$
    1. $C_1 : \forall x \in MWF_G \cup TR_G$, count of$(x) = s \leq$ gamemax(s), for $s \in$ Slots.
    2. $C_2 : \forall x \in MW_P \cup TR_P \cup F_P$, count of$(x) = s \leq$ practicemax(s), for $s \in$ Slots.
    3. $C_3 : \forall x = g_i$ for Division $i \in MWF_G \cup TR_G$, $\nexists x' = p_{ik_i} \in MW_P \cup TR_P \cup F_P$, such that assign$(x) =$ assign(x').
    4. $C_4 : \forall$ not compatible(a, b) where $x = a$, $x' = b$, $a, b \in Games \cup Practices$, assign$(x) \neq$ assign(x').
    5. $C_5 : \forall$ unwanted(a, s) where $a \in Games \cup Practices$ and $s \in Slots$, $\nexists x = a$ such that assign$(a) = s$.
    6. $C_6 : \forall x \in Games \cup Practices$, if Div of $x \geq 9$, assign$(x) = s$ such that s is in the evening.

7. $C_7 : \forall x \in Games$ where tier of x is U15/16/17/19, $\nexists x' \in Games,\ x' \neq x,$ and tier of x' is also U15/16/17/19.
8. $C_8 : \forall x \in X,\ \nexists$ assign$(x) = s$ such that $s = TR\ 11:00 - 12:30$.
9. $C_9 : \forall x \in X,\ x \in Games,$ if $(x = CMSA\,U12T1S \vee x = CMSA\,U13T1S) \rightarrow$ assign$(x) = TR\ 18:00 - 19:00$.
10. $C_{10} : \forall x \in X,$ if x is for $CMSA\,U12T1 \wedge x \neq CMSA\,U12T1S \rightarrow$ assign$(x) \neq TR\ 18:00 - 19:00$.
11. $C_{11} : \forall x \in X,$ if x is for $CMSA\,U13T1 \wedge x \neq CMSA\,U13T1S \rightarrow$ assign$(x) \neq TR\ 18:00 - 19:00$.
12. $C_{12} : \forall x = g_i \in Games,\ \forall x' = p_{ik}$,
assign$(x) = s_i,$ assign$(x') = s_j$ such that same time$(s_i, s_j) =$ false.

– If one of the pr_j is solved, perform the transition that changes its sol-entry. If there are several, select one of them randomly.

– Else if one of the pr_j is unsolvable, perform the transition that changes its sol-entry. If there are several, again select one of them randomly.

– Else, select the leaf(pr_j , ?) such that

1. $C_{\text{solved}}(pr_j) = \max_{prl}(\{C_{\text{solved}}(pr_l)\})$
2. If there are several, select the deepest leaf in the tree with this property.
3. If there are still several, select the one most left in the tree.

5.1.2 Or-tree-based Search Instance

- Search Instance: $Ins_\vee = (s_0, G_\vee)$
- $s_0 = \{partassign\}$
if the given problem to solve is pr, then we have:
  – $s = (pr', yes)$ or
  – $s = (pr', ?, b_1, ..., b_n), G_\vee(b_i) = yes$ for an i, or
  – All leafs of s have either the sol-entry no, or cannot be processed using *Altern*

6 Search Instance

The search instance of our model defines the initial state s_0 as an empty set, which will be passed into our model. The goal condition decides that the search terminates when we find

a solution that has a desired penalty score, $score_{desired}$, when we have applied the extension rule pop_{max} times, or when there's no more rules to apply in our model. The solution with the desired score is returned. For the case that a desired score is not reached, return the solution with the lowest penalty.

- **Search Instance:** $\text{Ins} = (s_0, G)$
- **Define** $s_0 : \ s_0 = \{\}$
- **Define** $s_{\text{goal}} : \ s_{\text{goal}} \in 2^F$ such that$\exists f_i \in s_{\text{goal}}$ where f_i such that

$$\text{Penalty}(\text{assigned}(GP, f_i)) \leq \text{Score}_{\text{desired}}$$

  $\text{Score}_{\text{desired}}$ is a predefined constant that will be determined during the implementation phase.
  This goal condition is in place to improve efficiency in the case that a desired solution is identified early.

- **Goal condition:** $G : S \to \{\text{yes}, \text{no}\}$

$$G(s_i) = \text{yes if and only if } s_{\text{goal}} \subseteq s_i, \text{ or}$$

$$\text{there is no extension rule applicable in } s_i, \text{ or}$$

$$\text{when we reach the final iteration of extension.}$$

- The final iteration of extension limits the number of times we populate new generations. This is predefined with a constant value $pop_{\text{max}} \in \mathbb{N}$.
- When we reach the final iteration, set the s_{goal} to contain the individual with the lowest penalty; if there are multiple, the left-most one.
  Let $s_{\text{goal}} = \{f\} \in s_i$ where $\nexists g \in s_i$ such that $\text{Penalty}(g) < \text{Penalty}(f)$