Basketball strategy design: a simulation approach

Guilherme Otranto [1], Leonardo Lamas [2], Junior Barrera [1]
1 Institute of Mathematics and Statistics - University of São Paulo
2 Faculty of Physical Education - University of Brasilia

Abstract

The goal of this study was to create a basketball game simulator capable of applying user defined strategies to guide the behavior of the agents in the simulation. For this purpose, we created a formal strategy model to describe complex team behavior and developed methods of using that model to calculate collective plans. We defined both the strategy model and the planning methods in a broad manner that can be applied in many different domains. Then we defined a basketball simulation domain and implemented our methodology. The resulting formal strategy model for basketball can be used to represent team behavior, analyze real world events and create simulations that indicate how different strategies perform against each other.

Introduction

Performance analysis in basketball has presented a fast increase in the past few years (Gudmundsson and Horton 2017). Benefited by progressively availability of game tracking data, acquired in quite high frequency for the purpose of use, these researches provide insights on individual (Lucey et al. 2014) and collective behaviors (Oh, Keshri, and Iyengar 2015) during games, contributing for predicting outcomes even in specific tactical circumstances of a ball possession (Cervone et al. 2016).

In this "big data" scenario, data-driven approaches proliferate and, eventually, some of them explore the control of players' behaviors, based on the data assessed during games (Felsen and Lucey 2017), referring to planning issues. In team sports, there is vast empirical literature dealing with planning, according to the so-called team strategy. Nonetheless, fewer scientific contributions have been given to this topic (Lamas et al. 2014a; Lennartsson, Lidström, and Lindberg 2015; Otranto 2017), which is central for enabling cooperation in every multi-agent competitive environment, including a basketball game (Otranto 2017; Lamas et al. 2014a). Hence, our initial goal was to create a realistic basketball game simulator that accepted the user's input as a guide to the collective team behavior.

The main challenge in creating a team sport simulator with user defined strategy is the complexity of the domain. In sports, there is an infinite number of different possible situations (i.e., domain states) where our simulator must be able to provide an action plan for each player. We would also like that plan to be consistent with some user defined strategy for every possible situation. To make this task possible, we created a method that simplifies the domain by aggregating similar situations into equivalence classes. This allows us to plan actions in a domain with a finite and known number of possible states and later translate those actions to the original domain. It also allows the user to provide meaningful strategies by specifying actions for each class of states in the simplified domain.

Our work is divided in two parts that will be described in this document. The first part details the model for simplifying a continuous and infinite domain into a discreet and finite domain. The second part details the methods that we can use to apply results acquired in the simplified domain to the original one. For any situation in the original domain we can extract an action play in three steps: 1 - identify a representative state in the simplified domain; 2 - extract a plan using the desired user defined strategy; 3 - adapt the actions of the plan for the particular situation of the original domain.

Formal Strategy Model

We developed a formal strategy model capable of describing a control system for a team of agents in complex environments. While our initial effort was aimed at using the strategy model to describe team strategy in invasion team sports environments (e.g., soccer, basketball, hokey), the resulting system is generic enough to be used outside this domain. In its essence, the strategy model creates equivalence classes of the domain's state space and thus allows for a compact representation of only the strategically significant data.

The formal strategy model creates a structured representation of the desired collective behavior of a team of agents collaborating towards some shared goal. It is a description language for any team strategy. We name a specific team strategy, described through our model, a Team Strategy Model (TSM).

Our model was created to control a team of agents in a cooperative dynamic stochastic system. The system can also be competitive, in which case each team will use its own TSM to control their actions. A domain is compatible with our formal model if it has:

- A *state space* S, which can be infinite and non-enumerable, where each state $s \in S$ can be described by a finite set of variables V.
- An *action space* $A(s)$ for each state $s \in S$.
- A non-empty set of *possible outcomes* O, $\forall a \in A(s)$, $\forall s \in S$. An outcome $o \in O(s, a)$ is a state of the state space S.
- For each $s, s' \in S$ and variable $v \in V$ such that only the value of v differs between s and s', there is an outcome $o \in O(s, a)$ for some action $a \in A(s)$ such that $o = s'$.

Note that both the state space and the action space can be infinite. The first due to continuous variables in V and the second by utilizing parametric actions. Any domain can be compatible with our model if: 1 - states can be described by a finite set of variables; 2 - we know at least one possible outcome for every action; 3 - we have a control action that can change the value of each variable. A TSM can be applied to domains where the state space is infinite due to the use of equivalence classes to restrict and group the state space to contain only relevant strategic situations.

Strategy Map

The strategy map (SM) is a representation of a class of relevant situations for some strategy defined in a TSM. The SM can group any number of states in the domain's state space into a single state in the TSM. Our formal model allows the SM to create equivalence classes in one of two ways: information omission or information grouping.

Information omission is analogous to the "don't care" variables used when creating intervals in boolean algebra (Hill and Peterson 1968). If a state can be described by a set of variables V and there is a subset $V_s \subseteq V$, then an equivalence class can be created by assigning values to every $v \in V_s$ while omitting all $v' \notin V_s$. Every state in the domain for which the values of every $v \in V_s$ coincide with the assigned values will be captured by this equivalence class, regardless of the values of the variables not in V_s. One could create an all encompassing class by setting $V_s = \emptyset$, which can be useful in error handling.

Information grouping defines discreet and finite intervals to describe continuous variables in a manner analogous to interval creation in continuous variables statistical analysis. The interval creation is done by delimiting the possible values of some continuous variable. Take, for instance, some $v \in V$ such that v is a real number ($v \in \mathbb{R}$), then we can define values $a, b \in \mathbb{R}$ and the interval $[a, b]$. Every state where $v \in [a, b]$ could be captured by the defined equivalence class. More broadly, any variable in V that belongs to a group with well defined order (e.g., $\mathbb{N}$, $\mathbb{I}$, $\mathbb{R}$) can be described by an equivalence class through this method.

For each variable in a domain, the SM can either create an interval or omit the variable completely. The SM also defines a weight for each variable, which signifies the relative importance of that constraint over the others. This weight is useful when measuring how closely a SM is describing some domain situation.

For a domain with set of variables V, a SM is formally defined as a set of constraints C, where each $c \in C$ has:

- An *associated variable* $v \in V$, such that $v \in G$ and G is an ordered set with order $<_G$.
- A *start* and an *end* bounds $a, b \in G$ such that $a <_G b$.
- A *weight* $w \in \mathbb{R}^*$.

Figure 1 illustrates the process of creating SMs in a domain with two continuous variables (v_1 and v_2). Both maps define a constrain for each domain variable. SM_1 constraints the variable v_1 with c_1 and the variable v_2 with c_2. SM_2 constraints the variable v_1 with c_1' and the variable v_2 with c_2'.

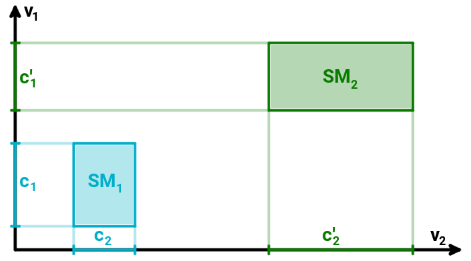

Figure 1: Two strategy maps (SM_1 and SM_2) defined in a continuous domain with two variables.

Strategy Transition

A strategy transition (ST) contains the necessary information to inform the system of how the transitions between SMs should occur in the TSM. A ST defines the actions that should be executed when transitioning between states. It would be impossible to specify valid actions to transition between every possible combination of the original domain states that are represented in the SMs, so the ST specifies domain actions necessary to transition between representatives states of each SM. While the defined actions might not accomplish the transition between all the combinations, it provides enough information to guide the transition using the methods described in this study. We define a ST as follows:

- An origin SM α.
- A destination SM β.
- Representative states $\alpha_r \in \alpha$ and $\beta_r \in \beta$.
- A list of actions $\{a_1, a_2, ..., a_n\}$ with $a_i \in A(s_i)$ and $s_i \in S, \forall i \in [1, n]$ such that:

  There is a list of states $\{s_1, s_2, ..., s_{n+1}\}$ with $s_j \in S$, $\forall j \in [1, n+1]$ where:
  $s_{x+1} \in O(s_x, a_x)$, $a_x \in A(s_x)$, $s_1 = \alpha_r$ and $s_{n+1} = \beta_r$, $\forall x \in [1, n]$

Note that we use the notation $\alpha_r \in \alpha$ to indicate that α_r is a state in the original domain that obeys every constraint defined in α (i.e., α_r belongs to the equivalence class defined by α). So, the ST selects a representative for each map and provides a list of actions that *may* take the system from one representative to the other. A list is valid as long as one of the possible outcomes of every action leads to the next step in the chain that eventually leads to the chosen destination.

It is important to note that $\{a_1, a_2, ..., a_n\}$ are not necessarily the most efficient way to transition between α_r and β_r. They simply represent what the designer of the strategy would like the agents to perform when they find themselves in SM_1 and wish to reach SM_2. The choice of actions, representatives and even SMs are up to the designer and our objective is to apply them faithfully, not necessarily to control the agents in the best possible manner.

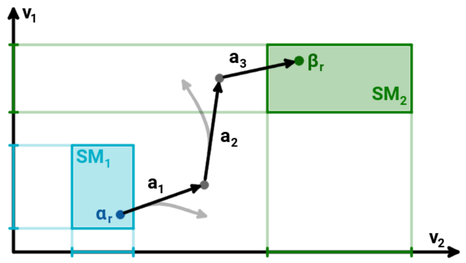

Figure 2: A strategy transition with actions $\{a_1, a_2, a_3\}$ and representative states $\alpha_r \in SM_1$ and $\beta_r \in SM_2$. Other outcomes of actions a_1 and a_2 have been faded to improve readability.

Strategy Graph

The strategy graph is a natural structure to represent the TSM: a directed graph with SMs as the nodes and STs as the edges. We use the graph to encode extra navigational information about the TSM. On each edge of the graph (i.e., for each ST) we encode a risk value that represents the probability of something going wrong during the transition and on each node (i.e., each SM) we encode a reward value representing how advantageous we consider the situations described by that SM to be. Usually, a planning problem will work with only one of those variables, but adapting the algorithm to work with both values is trivial and we found that such representation is more easily understood by the users.

If we know the probability distribution of the actions in the domain, we have a good candidate for the risk value of a ST: one minus the product of the probability of each action leading to the next state in the chain. This is not required, the risk and reward values encoded in the strategy graph can be provided by the user or learned by the system through simulations. Considering the risk to be the probability of a transition to fail and the reward to be the expected accumulated reward from that state after some time, we can estimate those values after several simulation rounds.

We define a strategy graph as:

- A set of states S, each $s \in S$ contains:
  - A SM m.
  - A reward value $\gamma \in \mathbb{R}^*$.
- A set of edges E, each $e \in E$ contains:
  - A ST t.
  - An origin state $\alpha \in S$.
  - A destination state $\beta \in S$.
  - A risk value $\omega \in [0, 1]$.

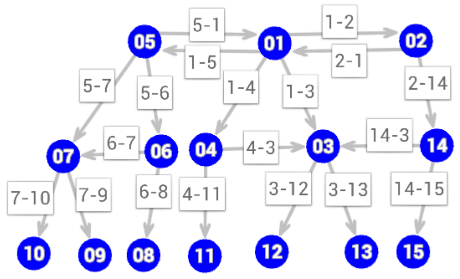

Figure 3: A simple strategy graph, the values for the risks and rewards have been omitted.

Figure 3 shows what a simple strategy graph might look like. The amount of states in the graph and the number of edges leaving each state are indicators of the complexity of the strategy encoded within. As we will discuss further on, our tests indicated that fairly complex behavior emerges with only a few dozen states in the strategy graph.

A graph contains the whole TSM of a team, but we do not need to use the whole model all the time. It is possible to define subsets of the graph to be used at different times, for instance, one could exclude some states and transitions that represent riskier behavior in order to apply a safer strategy. A *Finite State Machine* (FSM) can be defined to control which subset of the graph should be active at any given moment: the states of the machine would be graph subsets and the transitions would be changes in key control parameters. This sophistication allows the TSM to dynamically change the team's behavior based on high level changes in the environment.

Event analysis using the formal model

A direct result of creating a formal strategy model is that we can use it to analyze real world events that take place in the domain described by our model. For any domain where we can capture real world events and infer their corresponding state in the domain's state space, we can use a TSM to measure how well the event matches the model. We have so far used this technique in the domain of invasion team sports (Lamas et al. 2014b), where a state in the domain can be usually described by the position of all the players and the ball. Some studies have also attempted to infer the strategy from real world events (Lucey et al. 2012).

There are significant efforts in the sport scientific community to capture player's movements and positioning during matches. This information can be used to detect how closely a team is following some TSM defined for that domain. Our TSM works with equivalence classes so we can define very broad classes and use only approximate positioning information to analyze the match. We have worked with goalkeeper analysis where the match information required could be extracted by watching regular game video footage. The only information needed was the area in the field where the player with the ball was located and the goalkeeper's general position (Lamas et al. 2018).

Planning with the Strategy Model

By using the TSM in planning we can shift from a complex domain to a much simpler planning domain. This enables us to work with a finite, and often quite small, number of possible states for the planning. In this section we will show how we can apply plans calculated in the TSM to some state in the original domain. For clarity, we will call a state in the original domain a *situation*.

The objective of the planning process is: given some initial situation, calculate applicable actions that are consistent with some input strategy. One issue we need to deal with is that the situations and actions exist in a different domain than the one where the strategy is defined. In order to solve this we created 2 translation functions: one that maps a situation to a state in the strategy and a second that maps transitions in the strategy to actions in the original domain. Our approach to planning using a TSM requires 3 separate steps:

1. *Matching*: Translate a situation into a SM of the TSM.
2. *Planning*: Extract a plan from the TSM.
3. *Realization*: Adapt the plan so it fits the situation.

The Matching Function

The objective of the matching function is to find the SM in the TSM that best represents some situation. In order to do this, a metric is established that measures how well any given SM represents a situation. While it would be trivial to search for a SM whose equivalence class encompasses the situation, we have no guarantees that such a SM exists for every situation in the original domain. A metric solves this problem by allowing some error to be introduced to the matching process when required. This is similar to the nearest-neighbor approach in machine learning. As noted in (Blum and Langley 1997), such approach is very sensitive to what attributes are used in the metric. We follow their idea of defining weights for each variable used by the metric, thus providing a tool to remove irrelevant attributes.

Given a metric μ to measure distance between SMs and a given situation, the matching function is quite simple: simply return the SM in the TSM that minimizes the distance to the situation being matched. In order to define μ, consider the following:

- Let $\{v_1, v_2, ..., v_n\}$ be the set of all domain variables in V.
- Let $\{G_1, G_2, ..., G_n\}$ be groups with orders $\{<_{G1}, <_{G2}, ..., <_{Gn}\}$ such that $v_i \in G_i, \forall i \in [1, n]$.
- For each $i \in [1, n]$, let functions δ_i: $G_i^2 \to \mathbb{R}^*$ be such that $\delta_i(x, y)$ with $x, y \in G_i$ represent the distance between x and y in G_i.
- Let a situation be described by the variables $\{x_1, x_2, ..., x_n\}$.
- Let a SM be defined by:
  - A set of omitted variables $O = \{o_1, o_2, ..., o_k\}$.
  - A set of grouping intervals $I = \{q_1, q_2, ..., q_l\}$, with components $q_i.a$ and $q_i.b$ being the start and end bounds respectively.
  - Such that $k + l = n$.
- Without loss of generality assume $SM = \{O, I\} = \{c_1, c_2, ..., c_n\}$ and that c_i is the constraint of x_i, $\forall i \in [1, n]$.
- Let $\{w_1, w_2, ..., w_n\} \in \mathbb{R}^n$ be the weights of the constraints of SM.

With the above assumptions we are ready to define the metric μ to compare the situation described by $\{x_1, x_2, ..., x_n\}$ and a SM. We do this by creating sub-metrics $\{\mu_1, \mu_2, ..., \mu_n\}$ such that:

$$\mu_i = \begin{cases} 0 & \text{if } i \leq k \\ 0 & \text{if } q_i.a <_{Gi} x_i <_{Gi} q_i.b \\ min_i & \text{otherwise} \end{cases} \quad (1)$$

with:

$$min_i = min(\delta_i(x_i, q_i.a), \delta_i(x_i, q_i.b)) \quad (2)$$

Then we can finally define μ simply as:

$$\mu = \sum_{i=1}^{n} \mu_i w_i \quad (3)$$

For a variable $v \in G$, the distance function $\delta : G^2 \to \mathbb{R}^*$ can be any domain specific function that suits a need, but it is worth mentioning that the Euclidean distance works well when G is $\mathbb{I}^2$ or $\mathbb{R}^2$. When G is $\mathbb{I}$ or $\mathbb{R}$ the absolute value of the difference also works well. In any of the two cases, one might want to consider using the squared value of the result in order to penalize greater differences in favor of multiple smaller ones.

Figure 4 illustrates the matching function. Simply put, the matching function will calculate a distance between each variable in the situation's description and the corresponding constraint in the SMs. Then it calculates a weighted sum using those distances and selects the SM that minimizes the sum to represent the situation in the TSM.

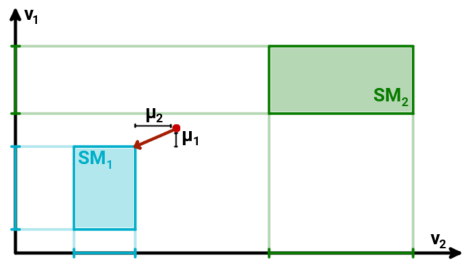

Figure 4: An example of the matching function. The situation marked in red is matched to SM_1 after μ_1 and $\mu2$ are calculated.

The Planning Function

The objective of the planning is to calculate a sequence of STs that starts in the SM found by the matching function. The risk and reward values in the strategy graph are used to calculate the ideal sequence of STs, which we will refer to as a *play*. So a play is a path in the strategy graph that begins in the SM that best describes the current domain situation, as calculated by matching. It would be trivial for this process to calculate a play that maximizes some constraint, such as reward divided by risk, specially due to the small size of the strategy graph. However, the objective is to extract different plays in a manner consistent with the TSM, resulting in a more realistic and unpredictable behavior.

We would like to explore the TSM in a manner that is consistent with the intent of the strategy designer. Therefore, we cannot always simply pick the play that maximizes some constraint, but we can use the constraint to prioritize our play selection. We do this by defining a *score* for each play and then we use this score to define a distribution over which we will randomly select the play we want. Ideally, the plays with lower risk and greater rewards in the TSM should be picked more often, while increased risks or lower rewards should decrease the chance of a play being selected. This is a balance between exploration (trying every play) and exploitation (using better plays more often). This approach borrows heavily from the literature on Monte Carlo Tree Search methods (Browne et al. 2012) and have been used successfully in domains where there is uncertainty or the domain is not fully observable.

Let's define a score function ψ for some play defined in a strategy graph with states S and edges E:

- Consider a play as $p = \{e_1, e_2, ..., e_n\}$, with $e_i \in E$, $\forall i \in [1, n]$ and:
  - $e_i.\beta \in S$ is the destination state of edge e_i.
  - $e_i.\beta.\gamma \in \mathbb{R}^*$ is the reward value of state $e_i.\beta$.
  - $e_i.\omega \in \mathbb{R}^*$ is the risk value of edge e_i.

Then ψ is a function $\psi{:}E^n \to \mathbb{R}^*$ calculated as:

$$\psi(p) = \sum_{i=1}^{n} (1 - e_i.\omega)\; e_i.\beta.\gamma \tag{4}$$

We use $(1 - e_i.\omega)$ because it has an intuitive meaning: if $e_i.\omega$ is the probability that a transition will fail, then $(1-e_i.\omega)$ is the probability that it will succeed. The function ψ manages to capture the reward associated with a play in the strategy graph as well as the probability of acquiring the reward. If the reward associated with any state is zero, or if the risk associated with any edge is one, then the value of ψ is zero. This might not be desirable, since it would mean the state (or edge) is useless in the strategy and so is any play using it. To avoid this we can introduce a small value $\delta \in \mathbb{R}^+$, $\delta \ll 1$ and use it to clamp the values in the calculations, like so:

$$\psi(p) = \sum_{i=1}^{n} max(\delta, (1 - e_i.\omega))\; max(\delta, e_i.\beta.\gamma) \tag{5}$$

This ensures that every play as a non-zero probability of being picked, which makes sense if we consider that: 1 - the play can be extracted from the strategy graph; 2 - the user designed the strategy graph to contain the play. We consider two methods of picking a play, the long-term and the short-term planning:

- Short-term planning: A play is assembled one edge at a time, starting at the initial state and adding edges until there are no more eligible edges. An eligible edge is an edge whose origin state is the current state and whose destination state is not yet part of the play. To pick the next edge we list every eligible edge and pick one as follows:
  - Let $\{e_1, e_2, ..., e_n\}$ be the set of eligible edges in E.
  - Let $\{\psi(\{e_1\}), \psi(\{e_2\}), ..., \psi(\{e_n\})\}$ be the score of the single edge plays defined above.
  - Let $T = \sum_{i=1}^{n} \psi(\{e_i\})$.
  - Pick an edge in a way that $P(e_i) = \psi(\{e_i\})/T$ is the probability of edge e_i being picked, $\forall i \in [1, n]$.
- Long-term planning: Every possible play that starts from the initial state is listed and one is selected as follows:
  - Let $\{p_1, p_2, ..., p_k\}$ be the set of possible plays.
  - Let $\{\psi(p_1), \psi(p_2), ..., \psi(p_k)\}$ be the scores of those plays.
  - Let $T = \sum_{i=1}^{k} \psi(p_i)$.
  - Pick a play in a way that $P(p_i) = \psi(p_i)/T$ is the probability of play p_i being picked, $\forall i \in [1, k]$.

A simple way to pick a play such that each p_i has the probability $P(p_i) = \psi(p_i)/T$ of being chosen is to uniformly pick a value $v \in (0, T]$ and choose the play p_i such that:

$$\sum_{j=1}^{i} \psi(p_j) < v \leq \sum_{j=1}^{i+1} \psi(p_j) \tag{6}$$

An analogous method can be used to select the next edge in the short-term play calculation. The difference between the methods of planning are indicative of the behavior we want to simulate. The short-term approach utilizes a greedy algorithm to create a play, which means that the immediate edges that are more attractive get selected more often. This could lead down a path with less attractive options, resulting in an overall worse play being selected more often than better options. This is not necessarily a bad thing, in some domains the greediness of this algorithm and the resulting quirk in the selection could be desirable traits for the simulation. If that is not the case, the long-term approach calculates an overall ideal distribution by comparing full plays. The difference in performance from both approaches can be ignored due to the small size of the strategy graphs.

The Realization Function

The *realization function* is responsible for ensuring that a play calculated by the planning function can be applied to the situation in the original domain and still accomplish the same behavior specified in the TSM. There are two parts of this adaptation that need to be considered, which we will call *corrections*:

1. The matching correction: This correction aims to fix the error introduced by the matching function. The amount of error introduced is measured by the μ metric defined in "The Matching Function".

2. The planning representative correction: This correction aims to fix the error introduced when we pick a single representative state α_r to represent a whole class of situations when defining a ST in "Strategy Transition".

There are several approaches that can be employed to make the necessary corrections, but they are variations between adapting the situation to fit the play or adapting the play to fit the situation. In order to adapt the situation we must add some actions to the beginning of the play so we can reach a more compatible state. In order to adapt the play we must change the actions within it so that they are compatible with the current situation.

Every adaptation made by the realization function uses the domain restriction mentioned in "Formal Strategy Model": there is always an action to control the value of a variable in V. In continuous domains this means that there are parametric actions to control the continuous variables. For instance: in a domain where there is a variable $v \in \mathbb{R}^3$ to control an object's position in space, there will be an action $a \in A(s)$, $\forall s \in S$ that changes that object's position to some arbitrary $x \in \mathbb{R}^3$. We will now formalize the process used to make each adaptation used by the approaches described above. We will use the same definition of a SM as we have in "The Matching Function".

There are 3 types of corrections that we use, and the following will describe how each correction can be accomplished:

1. Adapt $x \in S$ to a SM: for every variable $v \in V$ such that $v \in G$ with order $<_G$ and $x.v$ is the value of that variable in x:
   - If $v \in O$: nothing needs to be done.
   - If $v \in I$ and $q_v.a \leq_G x.v \leq_G q_v.b$: nothing needs to be done.
   - If $v \in I$ and $x.v \leq_G q_v.a$: create an action with outcome such that $x'.v = q_v.a$ when it is applied to x.
   - If $v \in I$ and $x.v \geq_G q_v.b$: create an action with outcome such that $x'.v = q_v.b$ when it is applied to x.
2. Adapt $x \in S$ to $y \in S$: for every variable $v \in V$ such that $v \in G$ with order $<_G$ and $x.v$ and $y.v$ are the values of that variable in x and y respectively: create an action with outcome such that $x'.v = y.v$ after it is applied to x.
3. Adapt $a \in A(y)$, $y \in S$ to $\{a_1, a_2, ..., a_n\} \in A(x)^n$, $x \in S$: This can be done in a relative manner (i.e., change both start and finish point of the action) or in an absolute manner (i.e., change only the start point of the action).
   - Relative: for every variable $v \in V$ such that $v \in G$ with order $<_G$ that is changed by a, create an action that creates this same change and apply it to x.
   - Absolute: for every variable $v \in V$ such that $v \in G$ with order $<_G$ that is changed by a, create an action such that $x.v = y.v$ after it is applied to x.

Results

As a result of the methods described above, we developed a formal strategy model applied to basketball with the goal of representing strategic knowledge in a structured manner that could be useful for both match analysis and strategy simulation by computer systems.

The Basketball's Strategy Map

The SM in basketball is a representation of a class of match situations idealized in the TSM defined by the user. In basketball a match situation can be defined by the positioning of each player and the position of the ball. We defined the variables in V for this domain as:

- A *position* in $\mathbb{R}^2$ for each player.
- A *velocity* in $\mathbb{R}^2$ for each player.
- A *rotation* in $\mathbb{R}$ for each player.
- A *ball holder index* in $\mathbb{N}$.
- A *ball position* in $\mathbb{R}^2$.
- A *ball velocity* in $\mathbb{R}^2$.

The SM creates what we call a *player role* for each match player in order to create the necessary equivalence classes using the concept of "playing area". The playing area is an elliptical interval we used to group the position and rotation information into an equivalence class. Furthermore, a SM does not need to specify every player role, omitted ones will simply enlarge the defined equivalence class to encompass any position and rotation for the omitted player. The player role can also indicate whether or not it has the ball. Every other variable in the domain is grouped by omission (i.e., ignored by the SM). Figure 5 shows an example of a SM where every player role was defined and player role 1 has the ball. Only the playing areas for the offensive team have been highlighted for simplicity.

Formally, we define a SM as a group of *player roles* P, where each player role $p \in P$ has:

- A team identifier $t \in \{\text{Offense, Defense}\}$.
- A two dimensional *center* $(x, y) \in \mathbb{R}^2$
- A *rotation* of the playing area $\alpha \in \mathbb{R}^2$. This is the angle between the positive x-axis and the facing of the playing area, counter-clock wise.
- A *rotation tolerance* $\alpha_t \in \mathbb{R}^*$.
- A *dimension* of the playing area $(w, h) \in \mathbb{R}^2$.
- A boolean value indicating *ball possession*.

It is not easy to create a set of SMs that cover every possible match situation, much less one that makes sense strategically. Fortunately such a set is not required in our proposed method. We created a matching function that allows any situation to be assigned to an existing SM, even if some error is introduced. So the specialist describing the SMs needs to consider only those that are important to the TSM being created. The number of SM necessary to describe complex strategies is quite small. Some of the traditional basketball strategies (e.g., triangles offensive), can be described with no more than a few dozen SMs. This small number of SMs greatly facilitates the handling of TSMs by computer systems, as well as facilitating the visualization by humans.

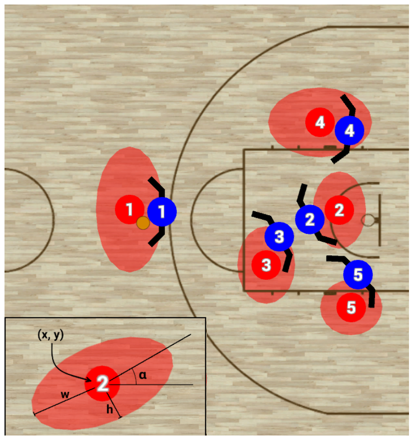

Figure 5: Strategy map with the playing areas highlighted in red for the player roles in the offensive team. The highlight on the lower left shows the variables that describe a single player role.

The Basketball's Strategy Transition

A ST allows the specialist to detail the necessary actions the players must execute in order to transition between SMs, these include: path and velocity of movement, passes, throws, idle wait times, cuts, etc. The timing of the actions (e.g., when the path of two player roles will cross) can be specified through a combination of movement velocity and idle wait times.

In order to choose the representative states for each SM, we consider the position of every player to be the center of the player role and we disregard the rotation tolerance. Any omitted player roles are put into predetermined positions outside the court. This SM representatives place each player in the mean position of every match situation encompassed by the SM. They also provide a natural representation of the transition when overlaid with the maps, since the movement starts and ends in the center of the playing areas.

We define a ST as follows:

- An origin SM α.
- A destination SM β.
- A set of paths P, one for each player role in α and β.
- Each path $p \in P$ contains a nonempty list of waypoints $W(p)$, each waypoint $w \in W(p)$ has:
  - An initial wait time $t \in \mathbb{R}^*$.
  - A start position $(x, y) \in \mathbb{R}^2$.
  - A movement speed $s \in \mathbb{R}^*$.
  - An end position $(x', y') \in \mathbb{R}^2$.
  - A final wait time $t' \in \mathbb{R}^*$.
- A set of passes B, each $b \in B$ has:
  - A start time $t \in \mathbb{R}^*$.
  - The path $p \in P$ that initiates the pass.
  - The path $r \in P$ that receives the pass.
- An optional shoot to basket time $t \in \mathbb{R}^*$.

A valid transition has a few requirements that need to be enforced when it is being created. To ensure that the ST contains an applicable set of actions whose execution can lead from α to β: 1 - the start position of the initial waypoints must coincide with their player role's center in α; 2 - the end positions of the last waypoints must coincide with their player role's center in β; 3 -each waypoint after the first must start at the previous waypoint's end position; 4 - for every pass, the player role that initiates the pass must have ball possession; 5 - the player with ball possession in the end must have the ball in β.

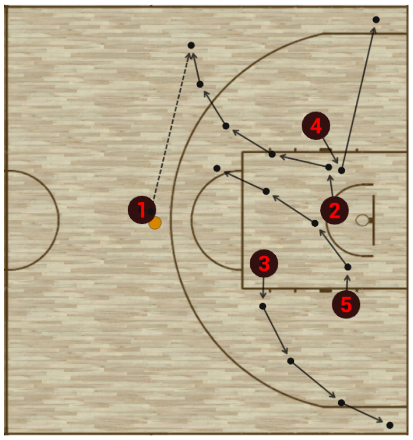

Figure 6: A strategy transition with a single pass from player 1 to player 2.

The Basketball's Strategy Graph

For the basketball's strategy graph we chose to interpret the risk as the probability of a transition failing due to some action not yielding the desired result. This means a pass failing due to an interception by the adversary, a fumble or a carry failing due to the ball being stolen. The reward of a state is simply the probability of the team scoring some points after being in that state. Both these values can be specified by the user and they could also be estimated though simulation after several rounds.

We define the strategy graph for basketball as:

- A set of states S, each $s \in S$ contains:
  - A state id.
  - A SM m.
  - A reward value $\gamma \in [0, 1]$.
- A set of edges E, each $e \in E$ contains:
  - A transition id.
  - A ST t.
  - An origin state $\alpha \in S$.
  - A destination state $\beta \in S$.
  - A risk value $\omega \in [0, 1]$.

A FSM to control which subset of the strategy graph is active during any time in the match is very useful to enhance the adaptability of the strategy. Each subset of the graph can hold only the appropriate plays for some match situation. In basketball, several match situations could be used to trigger a state change in the FSM, for instance: time left on the clock; match score; players on the court; current quarter being played. A designer can use this to create a specific set of plays that should only be executed in very specific scenarios.

Discussion

The TSM described in this document has allowed the creation of a complex strategy model for basketball that has already been used to describe complex strategies and it is currently supporting the development of a fully fledged basketball simulator.

It was fairly straight forward to implement *matching*, *planning* and *realization* functions for the basketball model. The resulting planner can always provide a simulator with a plan of relevant actions that fit the specified strategy. The actions provided by the planner follow the the strategy specifications while maintaining some situational awareness that may increase external validity of the simulation.

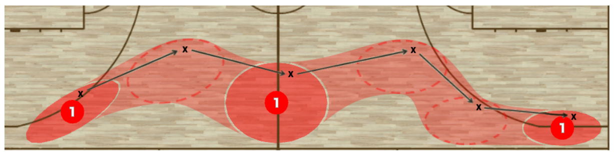

Figure 7: A realized plan for a single player (in black) and the corresponding path in the TSM (in red).

The game simulation method we have developed allows a specialist to specify his own basketball TSMs and verify how different strategies might react to each other. While the development of a complete match simulator is still in progress, a specialist can already visualize plays in motion and study different aspects of his TSM in a dynamic environment.

The TSMs that support the simulation procedures should be specified according to a formal language which guarantees systematic and accurate data input for the analysis of strategies structures. Therefore, besides empirical results obtained from teams' interactions along the simulation, it supports the investigation of relations between obtained performances and strategies' features. For instance, strategies presenting more derivation branches introduce greater unpredictability by providing alternative play options and favoring better rewards.

In conclusion, this study may indicate a promising research field of artificial intelligence applied to basketball, with several spin-offs for different sports and even other fields.

Acknowledgments

LL is supported by Fundacao de Amparo a Pesquisa do Distrito Federal:00193.00002099/2018-65 and Fundacao de Amparo a Pesquisa do Estado de Sao Paulo: 2015/01587-0. JB is supported by Fundacao de Amparo a Pesquisa do Estado de Sao Paulo: 2015/01587-0. This research was supported by FAPESP proc. 15/01587-0 and FAPESP proc. 14/50937-1.

References

Blum, A. L., and Langley, P. 1997. Selection of relevant features and examples in machine learning. *Artificial intelligence* 97(1):245–271.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games* 4(1):1–43.

Cervone, D.; D'Amour, A.; Bornn, L.; and Goldsberry, K. 2016. A multiresolution stochastic process model for predicting basketball possession outcomes. *Journal of the American Statistical Association* 111(514):585–599.

Felsen, P., and Lucey, P. 2017. Body shots: Analyzing shooting styles in the nba using body pose. In *MIT Sloan, Sports Analytics Conference*.

Gudmundsson, J., and Horton, M. 2017. Spatio-temporal analysis of team sports. *ACM Computing Surveys (CSUR)* 50(2):22.

Hill, F. J., and Peterson, G. R. 1968. *Introduction to switching theory and logic design*. Wiley.

Lamas, L.; Barrera, J.; Otranto, G.; and Ugrinowitsch, C. 2014a. Invasion team sports: strategy and match modeling. *International Journal of Performance Analysis in Sport* 14(1):307–329.

Lamas, L.; Santana, F.; Otranto, G.; and Barrera, J. 2014b. Inference of team sports strategies based on a library of states: application to basketball. In *Proceedings of the 2014 KDD Workshop on Large-Scale Sports Analytics*.

Lamas, L.; Drezner, R.; Otranto, G.; and Barrera, J. 2018. Analytic method for evaluating players' decisions in team sports: Applications to the soccer goalkeeper. *PloS one* 13(2):e0191431.

Lennartsson, J.; Lidström, N.; and Lindberg, C. 2015. Game intelligence in team sports. *PloS one* 10(5):e0125453.

Lucey, P.; Bialkowski, A.; Carr, P.; Foote, E.; and Matthews, I. A. 2012. Characterizing multi-agent team behavior from partial team tracings: Evidence from the english premier league. In *AAAI*.

Lucey, P.; Bialkowski, A.; Carr, P.; Yue, Y.; and Matthews, I. 2014. How to get an open shot: Analyzing team movement in basketball using tracking data. In *Proceedings of the 8th annual MIT SLOAN sports analytics conference*.

Oh, M.-h.; Keshri, S.; and Iyengar, G. 2015. Graphical model for baskeball match simulation. In *Proceddings of the 2015 MIT Sloan Sports Analytics Conference, Boston, MA, USA*, volume 2728.

Otranto, G. F. 2017. *A formal model for strategic planning in cooperative and competitive environments case study: design and implementation of a basketball simulator*. Ph.D. Dissertation, Universidade de São Paulo.