Initial Draft: A full market dynamics model

Mark Brezina

May 2025

1 Introduction

We formalize the system as follows. Let $t = 0, 1, 2, \ldots$ be discrete time steps. The environment consists of (1) a set of hierarchical **rules** governing dynamics and constraints, (2) a time-varying graph $G(t) = (V, E(t))$ whose *Vertices* represent moving objects (or "particles") in the system, (3) a discretized **geometrical surface** (e.g. a spatial lattice or mesh) on which values and fields evolve (modeled with multiple layers), and (4) a set of **agents** with states and utility functions. We describe each in turn.

- **Rules Hierarchy:** We assume a structured set of rules $\mathcal{R}$ (e.g. invariants, probabilistic transition rules, constraints) at multiple tiers. For example, Tier-1 rules might enforce conservation laws or feasibility constraints, Tier-2 rules govern probabilistic state transitions (e.g. if event E occurs, object state updates according to distribution $P(\cdot|E)$), and Tier-3 rules define outcomes of interactions among objects or agents. Such hierarchical, rule-based environments have been studied in agent-based modeling (e.g. Lotzmann & Meyer's declarative rule-based agent models). In general we let $\mathcal{R} = r_1, r_2, \ldots$ denote all rules; each rule r is a mapping from certain conditions (on agents, objects, or environment) to outcomes (possibly stochastic).
- **Graph Structure (Objects as Vertices):** We model abstract objects or particles as vertices in a directed graph. Concretely, let V be the set of graph nodes (for example, nodes can encode discrete space-time positions). An object moving from node i at time t to node j at time $t+1$ is represented by a directed vertice v from (i, t) to $(j, t+1)$. Thus each vertice encodes an object's trajectory between discrete space–time events (cf. "nodes = events, vertices = particles" in physical universe graphs). We denote the set of edges at time t by $E(t)$. Each edge (object) $e \in E(t)$ carries a state vector

$$\Theta_e(t) = (pos_e(t), val_e(t), \tau_e, ...)$$

  where $\mathrm{pos}_e(t)$ is its spatial position (or node location), $\mathrm{val}_e(t)$ its internal value or state, τ_e its type/category, etc. As the system evolves, edges can move, be created, or be destroyed according to the rules. This is analogous to interaction networks where nodes/edges represent objects and relations
- **Discretized Geometrical Surface and Layers:** The underlying spatial domain is discretized (e.g. a lattice or triangular mesh) into nodes $i \in V_{\text{space}}$ with coordinates x_i. On this surface we define multiple "information" layers. For example:
  - **Evaluation Layer** (E): A scalar field $E_i(t)$ at each site i representing local evaluation or utility density. (This could encode resource levels, reward potential, or other signals.)

  - **Propagation Layer (P)**: Encoded by a propagation matrix or adjacency on the grid (e.g. weights P_{ij}). This governs how values diffuse or propagate between neighboring sites (analogous to a graph Laplacian)
  - **Interaction Layer (I)**: A layer that captures how agents/objects at neighboring sites interact (e.g. collision effects or communication).

  These layers together form a multilayer network on the same surface nodes, with each layer representing a different aspect (evaluation, diffusion, interaction). Multilayer networks (networks with multiple types of relations or layers) are a standard tool for complex systems

- **Agents:** Let $\mathcal{A} = 1, \dots, N$ be the set of agents. Each agent i has a state $s_i(t)$ (which may include location on the surface, internal variables, etc.) and an action space $\mathcal{A}_i$. Agents follow: (a) General rules (e.g. physical laws or system-wide constraints) and (b) Agent-specific rules (e.g. strategic preferences or class-based behaviors). Each agent has a utility (or reward) function $U_i(t)$, which they seek to maximize (or loss L_i to minimize). Agents may act individually or form teams. The influence between agents is modeled by a time-varying interaction graph: for example, a weighted adjacency $W(t) = [w_{ij}(t)]$ between agents where $w_{ij}(t)$ is stochastic and time-varying. Such time-varying interaction networks are studied in network game theory. In particular, we can treat each pair (i, j) with weight $w_{ij}(t)$ as a (possibly random) communication or influence link (the graph of agent interactions may change each step). In summary, agents i have utility dynamics and interact on graphs as in multi-agent dynamic games.

2 Time-Evolving Equations

We now describe the discrete-time dynamics. Let the global state at time t be $X(t) = (S(t), E(t), s_i(t))$, where $S(t)$ denotes all surface layer values, $E(t)$ all object-edge states, and $s_i(t)$ agent states.

- **Object/Edge Dynamics:** Each edge (object) $e \in E(t)$ evolves according to rules. For example, if e moves from site i to j, its position updates as

$$\mathrm{pos}_e(t+1) = \mathrm{pos}_e(t) + \mathrm{v}_e(t)$$

  where $v_e(t)$ is a (possibly random) displacement determined by rules or physics. In general, the internal state $\Theta_e(t)$ follows a discrete update:

$$\Theta_e(t+1) = f_e(\Theta_e(t), \{\Theta_{e'}(t) : e' \sim e\}, S(\cdot, t), \xi_e(t))$$

  where f_e is a (possibly nonlinear) function encoding the rules, $e' \sim e$ are neighboring edges or objects, and $\xi_e(t)$ is noise. Concretely, one might write

$$val_e(t+1) = val_e(t) + \Delta t \cdot g_e(val_e(t), pos_e(t), S(pos_e(t), t)) + \eta_e(t)$$

capturing internal growth or decay. The appearance/disappearance of edges is likewise rule-driven: e.g., a collision rule may remove two edges and create a new one, according to a rule in $\mathcal{R}$.

- **Surface (Field) Dynamics:** Let $E_i(t)$ be the evaluation-layer value at site i, and let L be the discrete Laplacian (graph Laplacian) for the surface grid. Then a diffusion-like update is natural:

$$E(t+1) = E(t) + \Delta t(DLE(t) + F_{source}(t))$$

Here D is a diffusion coefficient, and $F_{\text{source}}(t)$ sums contributions from objects and agents. More explicitly, for each site i one can write:

$$E_i(t+1) = E_i(t) + \Delta t \left(D \sum_{j \in \mathcal{N}(i)} (E_j(t) - E_i(t)) + \sum_{e:e \; at \; i} \alpha_e(t) + \sum_{k:k \in \mathcal{A}_i(t)} \beta_{i,k}(t) \right)$$

where $\mathcal{N}(i)$ are neighbors of i, each object e at i injects amount $\alpha_e(t)$ into the field (e.g. resource deposit), and each agent k at site i injects $\beta_{i,k}(t)$ (e.g. information or consumption). The term $\sum_j (E_j - E_i)$ is the discrete Laplace operator on the grid (cf. In other words, surface values diffuse and receive source terms. (Similar equations govern other layers if needed, e.g. propagation weights may adapt, etc.)

- **Agent Dynamics:** Each agent i has state $s_i(t)$ and chooses action $a_i(t) \in \mathcal{A}i$. *The agent's state then updates by*
  $s_i(t+1) = G_i(s_i(t), S(\cdot, t), \{s_j(t)\}_{j \neq i}, a_i(t), \{a_j(t)\}_{j \neq i}),$
  according to general and agent-specific rules. For example, an agent's position may change on the surface, its wealth may increase by consuming resources (captured by S), etc. Its utility $U_i(t)$ is updated by an instantaneous reward (or loss):
  $U_i(t+1) = U_i(t) + r_i(s_i(t), S(\cdot, t), a_i(t), \{a_j(t)\}_{j \neq i})$
  where r_i is the reward function (negative of a loss) as defined by the agent's objectives. This covers agent–environment interactions and agent–agent interactions. In particular, agents may share information or constraints via the interaction graph weights $wij(t)$:
  for instance the reward r_i might depend on a weighted sum of neighbors' states $\sum_j w_{ij}(t) s_j(t)$, capturing communication or influence (with $w_{ij}(t)$ stochastic and time-varying

Collecting the above, the overall state $X(t)$ evolves as a (generally stochastic) dynamical system under the rule set $\mathcal{R}$:

$$X(t+1) = T\left(X(t), a_1(t), \ldots, a_N(t), \zeta(t)\right)$$

where $\zeta(t)$ represents random disturbances. This discrete-time system can be viewed as a high-dimensional Markov chain (or game, if multiple agents choose actions).

3 Optimal Policy for a Fully Informed Agent

We now outline how a single *fully informed* agent can compute an optimal policy (maximizing cumulative utility). If agent i observes the global state $X(t)$ and knows all rules, one can formulate its decision problem as a Markov Decision Process (MDP). Let the agent's *value function* $V_i(X, t)$ be the maximum expected future utility starting from state X at time t. The Bellman optimality equation gives:

$$V_i^*(X, t) = \underset{a_i \in \mathcal{A}_i}{max} \left\{ r_i(X, a_i) + \gamma \mathrm{E}\left[V_i^*(X', t+1) \mid X, a_i \right] \right\}$$

where $\gamma \in (0, 1]$ is a discount factor, $r_i(X, a_i)$ is the immediate reward (which may depend on X and a_i), and the expectation is over the next state X' distribution given (X, a_i) (others' actions can be folded into the environment or assumed known). In the undiscounted, finite-horizon case one sets $\gamma = 1$ up to horizon T. This equation is the discrete-time Bellman optimality condition, which embodies *backward induction* for dynamic optimization (cf. Bellman's principle).

In practice, one solves this by dynamic programming. A simple backward-induction algorithm (pseudocode) is:

```
Initialize: for all terminal states X, set V[T+1,X]=0.
For t = T down to 0 do:
    For each possible global state X:
        For each admissible action a_i of agent i:
            Compute immediate reward r = r_i(X, a_i).
            Compute expected next-state value:
                V_next = 0
                for each possible next state X':
                    V_next += P(X'|X, a_i) * V[t+1, X']
            TotalVal = r + gamma * V_next
        End for
        Choose a_i that maximizes TotalVal.
        Set V[t, X] = max_a_i TotalVal.
        Record policy pi(t,X) = argmax_a_i TotalVal.
    End for
End for
Return policy pi(.).
```

In words: at each time step t and state X, the agent evaluates each action a_i by summing immediate reward plus discounted future value (using transition probabilities $P(X'|X, a_i)$). It then picks the action maximizing expected utility. The optimal policy $\pi(t, X)$ and value function $V[t, X]$ are computed by this value-iteration (backward) procedure. In the infinite-horizon case, one can use the Bellman update iteratively until convergence. The correctness of this procedure follows from Bellman's optimality principle.

This abstract framework and algorithm are general and can incorporate any specific choices of rules, graph structure, surface discretization, and agent utilities. By clearly defining the state space and transition rules, one can systematically derive the difference equations above and compute policies, ensuring extensibility to varied applications in physics, economics, or game-theoretic domains.