**Parallel Population Protocol based on Dynamic Interactions and Equilibrium Computation in the Population Protocol Model**

**Student:**

Asaf Shnaider 316468636

Ofir Ofek 312168347

**Articles names:**

Game Dynamics and Equilibrium Computation in the Population Protocol Model

Dynamic Size Counting in the Population Protocol Model

**GitHub Link:** <https://github.com/AsafSchneiderman/Parallel-population>

**Introduction**

Population protocols provide a theoretical framework for modeling distributed systems of simple, memory-limited agents that interact randomly in pairs. These protocols are widely applicable in multi-agent systems, sensor networks, biological computing, and decentralized AI. Two recent papers explore different aspects of population protocols—one focusing on game dynamics and equilibrium computation and the other on dynamic population size estimation and synchronization.

The first paper, "**Game Dynamics and Equilibrium Computation in the Population Protocol Model**," introduces a new type of equilibrium—Distributional Equilibrium (DE)—which models how strategies evolve in a randomly interacting population. It analyzes the convergence properties of strategic decision-making, particularly in the context of repeated Prisoner’s Dilemma games, using random walk models. The study highlights key trade-offs between convergence speed, strategy complexity, and equilibrium quality, which are crucial for designing adaptive and decentralized AI agents.

The second paper, "**Dynamic Size Counting in the Population Protocol Model**," tackles the challenge of estimating the size of a dynamic population, where agents may join or leave over time. It introduces a loosely stabilizing size counting algorithm that provides an approximate log(𝑛) estimate, enabling synchronization mechanisms such as phase clocks. The approach ensures that distributed systems remain functional even when the number of agents changes unpredictably.

**Summary**

**Paper 1: Game Dynamics and Equilibrium Computation in the Population Protocol Model**

This paper introduces game-theoretic dynamics into population protocols, where agents update their strategies based on local payoffs from interactions.

**The main contributions are:**

1. **New Equilibrium Concept - Distributional Equilibrium:**

* Unlike traditional Nash equilibrium, Distributional Equilibrium models the stability of agent strategies in large, randomly interacting populations.
* It captures how individual strategy distributions settle into a steady state.
* The paper introduces the notion of an ϵ-approximate distributional equilibrium, in which the distribution µ over strategies is such that no single agent can individually change its strategy to improve its expected payoff by more than an ϵ margin.
* Steady state refers to a distribution over the agents' strategies that, once reached, remains (approximately) unchanged under the local update dynamics.
* Equilibrium (steady state) is reached when the evolution of the strategy distribution ceases to produce significant changes over time, despite ongoing pairwise interactions.

1. **Analysis of Repeated Prisoner’s Dilemma in Population Protocols:**

* The paper studies how cooperation and defection evolve in a multi-agent Prisoner’s Dilemma\* setting.
* The multi-agent setting is modeled via the population protocol framework, where many agents interact in pairs selected uniformly at random.
* Each interaction consists of a two-player prisoner's dilemma game – more precisely, a repeated version often cast as a "repeated donation game".
* In this version, after each round the game continues with a fixed probability δ (the continuous probability), allowing the same pair (or different pairs over time) to engage in a series of rounds.
* Instead of single pairs of players, the focus is on a population where many agents update their strategies through local, pairwise interactions.
* The game is played repeatedly (with each round having a chance to continue), which means that an agent's payoff is the aggregate over multiple rounds. This repetition enables strategies to evolve over time based on past interactions.

\*The Prisoner’s Dilemma is a classic game theory problem where two players must independently choose to cooperate (C) or defect (D) without knowing the other’s choice. While mutual cooperation yields a moderate reward for both, defecting provides a higher individual payoff if the other cooperates—but if both defect, they receive the lowest outcome, highlighting the tension between individual and collective rationality.

* Agents use local update rules based on their game payoffs to refine strategies.
* In the studied model, the population is divided into three fixed strategy types:
  + Always Cooperate (AC)
  + Always Defect (AD)
  + Generous Tit-For-Tat (GTFT)
* While AC and AD agents do not change their strategy, the update dynamics are applied to the GTFT agents.
* These agents hold a generosity parameter (g) that is discretized into a finite set *G = {g1, g2,…,gk}.*
* Their local update rule (named the k-IGT, explained below) adjusts this parameter *G* upward or downward based on the payoff they receive from their interaction partner.

1. **Mathematical Analysis Using High-Dimensional Random Walks:**

* Proves that the strategy distribution converges to a stable equilibrium.
* Uses Ehrenfest random walks to model mixing times (how fast equilibrium is reached).
  + That means that log(𝑛) evolution of the distribution over GTFT agents' generosity levels can be exactly mapped to a high-dimensional, weighted Ehrenfest process – a generalization of the classical two-urn Ehrenfest model. In this mapping:
    - Each "urn" corresponds to one of the discrete generosity values in the set *G*.
    - Transitions between urns (i.e. updates to the generosity parameter) occur with probabilities that depend on the type of interaction (whether the opponent is an AC/GTFT agent or an AD agent)
  + This equivalence allows the authors of the paper to analyze the mixing time (the number of interactions required for the distribution of generosity values to approach its stationary (steady state) multinomial distribution. The proof proceeds by:
    1. Showing that the k-IGT dynamics' transition exactly matches those of (k, a, b, m)-Ehrenfest process.
    2. Deriving the stationary distribution via detailed balance equations.
    3. Using coupling arguments and bounding techniques (such as spectral analysis) to establish upper and lower bounds on the mixing time.
* Demonstrates trade-offs between agent memory, convergence speed, and strategy complexity.

1. **development of k-IGT (Incremental Generosity Tuning) dynamics**:

* A local strategy update rule for agents using a GTFT (Generous Tit-for-Tat) strategy:
  + GTFT is a variant of the classical tit-for-tat strategy used in repeated prisoner's dilemma games.
  + In the standard approach, an agent simply mimics its opponent's previous action.
  + However, GTFT introduces a level of generosity by occasionally cooperating even after the opponent defects.
  + This forgiving aspect helps overcome issues like mutual retaliation caused by errors or noise in the game.
  + In the model, each GTFT agent's behavior is characterized by a generosity parameter *g* (the higher the value – the higher the forgiveness) which is adjusted via the k-IGT dynamics.
* Proves that k-IGT dynamics lead to an approximate distributional equilibrium with convergence guarantees and highlights the trade-off between memory complexity and convergence time:
  + K-IGT is a family of local update dynamics applied specifically to GTFT agents.
  + The continuous range of possible generosity values is discretized into *k* levels *{g1, g2, …, gk}*.
  + After each interaction, a GTFT agent adjusts its current generosity parameter by moving one level up (if the interaction was favorable, e.g. when facing cooperation) or one level down (if the outcome was poor, e.g. when facing defection).
  + The theoretical result shows that the k-IGT dynamics converge (in mixing time *O (k* *· n* *· log(n))* to a stationary distribution over the *k* generosity levels.
  + The normalized mean of this stationary distribution forms an **ϵ-approximate distributional equilibrium** with *ϵ=O(1/k)*
  + This means that by increasing *k*, the system can achieve a close approximation to a true equilibrium – but this comes at the cost of longer convergence times and higher memory requirements per agent (since each must store more possible states).

**Applications & Implications:**

* Multi-agent learning & AI: Helps in understanding adaptive behavior in distributed AI.
  + Adaptive behavior – The results show that even with limited memory and only local interaction, agents can adjust their strategies in a way that eventually leads to a stable, near-optimal distribution. This is valuable for designing systems where centralized control is impractical.
  + Cooperation over competition – In many distributed systems, agents are required to collaborate rather than engage in adversarial competition. The equilibrium analysis (via distributional equilibrium) demonstrates the cooperation can emerge naturally from simple, local update rules even when each agent acts in its self-interest.
  + The trade-offs highlighted provide insight into how to balance these factors when engineering distributed AI systems. This helps in designing protocols where agents learn and adapt in a cooperative setting without requiring expensive global coordination.
* Game theory & economics: Provides insights into cooperative behavior in decentralized systems.
* Networked systems: Useful for peer-to-peer networks, swarm robotics, and IoT.

**Paper 2: Dynamic Size Counting in the Population Protocol Model**

This paper introduces a size estimation algorithm for dynamic populations where agents may enter and leave arbitrarily. The goal is to provide an approximation of log(𝑛) where 𝑛 is the population size to enable more adaptive and synchronized population protocols.

**The main contributions are:**

1. **Dynamic Size Counting Algorithm:**

* Agents estimate log(𝑛) using geometrically distributed random variables.
* The algorithm adapts when the population changes, ensuring a stable estimate over time.
* The algorithm is a dynamic size counting protocol that uses GRVs (geometrically distributed random variables) to estimate the population size:
  + Each agent repeatedly "flips a coin" until it obtains a head, thereby sampling a GRV from a *Geom(1/2)* distribution.
  + Because the maximum value among 𝑛 independent GRVs is Θ(log(𝑛)) with high probability, agents exchange and update their maximum observed GRV via epidemic-style (gossip) spreading.
  + In addition, each agent maintains a timer ('time' variable) that is set proportional to its current maximum value.
  + When the timer expires (the system 'resets'), agents may sample a new GRV if it is larger than their current maximum.
  + This cyclical process adapts to changes in the population while ensuring that the estimate (the maximum GRV) is a constant-factor approximation of log(𝑛).

1. **Loosely Stabilizing Protocol for Dynamic Populations:**

* Unlike previous protocols, it remains valid for polynomial time before requiring recalibration.
* Previous size counting protocols typically assumed static populations or used leader-based or token-balancing approaches. For example, some earlier methods would propagate a size estimate based on coin flips or leader election, but these methods often required recalibration quickly (typically after a polylogarithmic or logarithmic, number of parallel time steps) since any change in population size (or the loss of a designated leader) could render the estimate inaccurate.
* In contrast, the dynamic protocol in this paper is designed to be loosely stabilizing. It converges from any arbitrary configuration to O(log(𝑛)) (or O(log(s)+log(n), where s is related to initial values) parallel time and then remains correct for polynomial number of interactions with high probability.
* This improved stability means that even when agents join or leave, the protocol holds a valid approximation for a much longer period before any recalibration is needed.
* Ensures stability even when agents leave or join dynamically, under continuously updating and propagating the maximum GRV among agents:
  + Epidemic spreading of the maximum – agents continuously share their current maximum GRV value with others. This "gossip" ensures that sufficiently high estimate is eventually adopted by almost all agents.
  + Phase-based operation – the protocol divides its operation into three phases (exchange, hold and reset) using a countdown timer (the phase clock, explained below).
  + Loosely-stabilizing design – once the system converges to a state where every agent holds a constant-factor approximation of log(𝑛), it remains in this correct configuration for a polynomial number of interactions. This stability is proven even in the presence of an adversary that can arbitrarily add or remove agents.
* Thus, through periodic resets coordinated by the phase clock and robust epidemic propagation, the protocol adapts to dynamic changes while maintaining an accurate size estimate.

1. **Uniform Phase Clock for Synchronization:**

* Uses size estimation to implement a loosely stabilizing phase clock:
  + A phase clock is a mechanism that organizes time into synchronized intervals (phases) without relying on a centralized clock. Each agent maintains a local timer, which is initially set as a multiple of its current maximum GRV (which serves as a proxy for log(𝑛)).
* Allow agents to remain synchronized, enabling coordination in distributed systems:
  + The phase clock divides the agent's time into three phases:
    - Exchange Phase – Agents exchange their current maximum GRV values.
    - Hold Phase – Agents maintain the current estimate if no larger value is encountered.
    - Reset Phase – When an agent's timer expires, it "resets" by sampling a new GRV and potentially updating its maximum.
* The duration of each phase is proportional to the current maximum. By doing so, the phase clock implicitly uses the size estimation (the GRV maximum) to determine phase lengths. Moreover, when agents reset almost simultaneously (due to the structure phase clock with overlapping 'burst' and 'overlap' intervals), they re-synchronize their estimates.

**Applications & Implications:**

* Biological Computing: Helps model cellular and chemical reaction networks.
* IoT & Sensor Networks: Supports self-organizing distributed systems.
* Swarm Robotics & Multi-Agent AI: Useful in environments where the number of agents fluctuates.

**Common Ground**

Both papers focus on population protocols model in which agents interact randomly in pairs. In *Game Dynamics and Equilibrium Computation in the Population Protocol Model*, the focus is on how local, pairwise interactions lead to the evolution of strategic behavior and convergence to an approximate distributional equilibrium in repeated games. Conversely, *Dynamic Size Counting in the Population Protocol Model* concentrates on how agents collaboratively estimate the population size (even as agents join or leave dynamically) by exchanging local information to approximate log(𝑛). Despite their different primary objectives, a key similarity is that both involve local interactions that lead to global convergence (one paper uses strategies updates to reach an equilibrium distribution of strategies, while the other employs epidemic-style propagation of locally computed random variables to maintain a consistent size estimate). Both also emphasize trade-offs between memory, time complexity, and accuracy in achieving their goals, or in other words, how global coordination and accuracy emerge from decentralized, pairwise interactions.

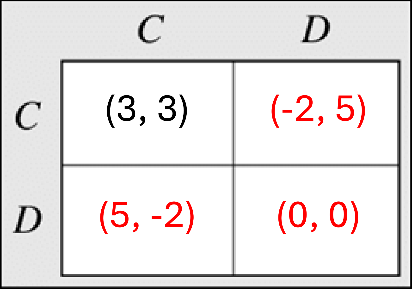
A crucial connection between these two works lies in the fact that strategy evolution in population protocols (Paper 1) is often dependent on knowing the number of agents in the system, which Paper 2 addresses with its dynamic size estimation. Our assumption is that by integrating adaptive population size estimation into equilibrium computation, it is possible to design more robust, scalable, and self-adjusting multi-agent systems that can handle real-world challenges such as fluctuating populations and decentralized learning.

Both protocols rely on local update rules and epidemic-style propagation to overcome the inherent limitations of decentralized computation. They provide rigorous analyses showing that even with very limited individual capabilities, collective behavior can be both robust and efficient.

**Our code**

In our code, we present our own suggested sequential population protocol algorithm that simultaneously addresses both game dynamics and dynamic size counting. The sequential example below is our idea (not taken from the papers) and is the basis of what we try to improve later using parallel programming. It is described as follows:

* Consider a population of N agents playing a Prisoner’s Dilemma game **variant** – a **donation game**, where each agent can either pick to be one of three strategies – Always Cooperate (***AC***), Always defect (***AD***) or Generous Tit-for-Tat **(GTFT)** either cooperate or defect according to their current generosity level– based on the following payoff matrix:



* Initially, the population starts as fixed proportions of agents of each strategy group.
* At each step, two randomly selected agents are picked to play the donation game.
* Both agents update their score based on the payoff matrix, and GTFT agents adjust their generosity levels using simplified k-IGT rule.
* All agents will update their dynamic size estimate by exchanging their geometric random variable (GRV) values, with a periodic reset mechanism to prevent immediate convergence.
* The simulation runs for M number of iterations; although the convergence of dynamic variables typically occurs earlier when run in a sequential (1-thread) mode.

**Suggested Improvements:**

* We propose an improved parallel version of the protocol by splitting the work of selecting agents and executing interacting between them across different numbers of threads - (e.g., 1, 2, 4, 8, 16, 32, 64, 128).
* Interaction & update of agents should be considered atomic for linearization point, so we implemented it using lock-free concurrency via atomic variable.

**Linearization point:**

The linearization points in our final implementation occur at the atomic update operation:

1. When an agent’s score is updated using an AtomicInteger.
2. When a GTFT agent’s generosity level is updated via the atomic updateAndGet method.
3. When the dynamic size count (the maxGRV) is updated atomically.

Thus, because these operations are lock-free and performed concurrently, each interaction is effectively linearizable without a global lock.

**Correctness Criteria**

A concurrent algorithm is **correct** if:

1. **Safety**: Atomic operations guarantee that no two threads modify shared data inconsistently.
2. **Liveness**: The lock-free design and the use of concurrent data structures ensure that the simulation continuously progresses without deadlocks or starvation.
3. **Consistency**: The concurrent (and atomic) updates of scores, generosity levels, and dynamic size estimates are linearizable, ensuring that the simulation’s overall outcome is consistent with the sequential specification.

**Benchmarking Our Solution:**

the output of our simulation (mpp.out), which is divided into two main sections: Agent Population Strategy Convergence & Population Size Estimation, and Execution Time Benchmarks.

***Part 1 – Agent Population Strategy Convergence & Population Size Estimation***

**Overview:**

* **Initial Setup:**  
  Each simulation run starts with 30% AC, 30% AD, and 40% GTFT agents. Given that the GTFT agents are split into 20% Cooperators and 20% Defectors, the initial distribution of cooperators and defectors is balanced at 50% each.
* **Periodic Updates:**  
  Every 100 iterations, the simulation logs the current percentages of agent types (Cooperators and Defectors) as well as an estimated population size based on a count-sizing mechanism.

**Analysis:**

* **Strategy Dynamics:**  
  The data shows that, within the first 100 iterations, the balance shifts from an even 50%-50% split to approximately 55% Cooperators and 45% Defectors. Over time, this gradually converges to around 65% Cooperators and 35% Defectors, with minor fluctuations observed between different runs, which means that the number of threads does not impact the convergence.
* **Population Size Estimation:**  
  The estimated population size, derived from dynamic count sizing, rapidly stabilizes.

***Part 2- Execution Time Benchmarks***

**Overview:**

* A graph with a line graph and numbers

  AI-generated content may be incorrect.**Measurement:**  
  The output file records the execution time (in milliseconds) for the simulation across various thread counts (1, 2, 4, 8, 16, 32, 64, 128). The following graph illustrates the average runtime of over five runs for each thread configuration:

**Observation:**

1. Sequential vs. Parallel Execution:
   * The result for 1 thread represents the baseline sequential (single-threaded) execution.
   * All other results reflect the benefits of parallel execution with varying numbers of threads.
2. Performance Trends:
   * Execution time drastically decreases from 1 to 2 threads, showing a significant performance gain.
   * As the number of threads increases to 4, 8, and 16, execution time continues to decrease but at a slower rate.
   * At 32 threads, execution time starts increasing slightly, suggesting overhead from thread contention or context switching begins to offset the gains.
   * At 64 threads, execution time decreases again but then increases back at 128 threads.
3. Interpretations:
   * The improvement from one to two threads underscores the effectiveness of parallelization.
   * The slight performance dip at 32 threads indicates a threshold where the overhead of managing many threads starts to impact performance.

\* We decided to only verify the convergence time, so we excluded the graph and results for checking the number of iterations.

**References:**

1. Kaaser, Dominik, and Maximilian Lohmann. "Dynamic Size Counting in the Population Protocol Model." *arXiv preprint arXiv:2405.05137* (2024).
2. Alistarh, Dan, et al. "Game Dynamics and Equilibrium Computation in the Population Protocol Model." *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing*. 2024.‏
3. Ben-Nun, Stav, et al. "An O (log3/2 n) parallel time population protocol for majority with O (log n) states." *Proceedings of the 39th Symposium on Principles of Distributed Computing*. 2020.‏
4. D. Ashlock, E. -Y. Kim and W. Ashlock, "A fingerprint comparison of different Prisoner's Dilemma payoff matrices," Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games, Copenhagen, Denmark, 2010, pp. 219-226, doi: 10.1109/ITW.2010.5593352.
5. Akçay, Erol, and Joan Roughgarden. "The evolution of payoff matrices: providing incentives to cooperate." *Proceedings of the Royal Society B: Biological Sciences* 278.1715 (2011): 2198-2206.
6. Eisert, Jens, and Martin Wilkens. "Quantum games." *Journal of Modern Optics* 47.14-15 (2000): 2543-2556.‏