Topics in Reinforcement Learning: AlphaZero, ChatGPT, Neuro-Dynamic Programming, Model Predictive Control, Discrete Optimization

Arizona State University

Course CSE 691, Spring 2024

Links to Class Notes, Videolectures, and Slides at
http://web.mit.edu/dimitrib/www/RLbook.html

Dimitri P. Bertsekas (dbertsek@asu.edu)

Yuchao Li (yuchaoli@asu.edu) Jamison Weber (jwweber@asu.edu)

Video Credit: Alejandro P. Riveiros William Emanuelsson Pratyusha Musunuru Dhanush Giriyan Devendra Parkar

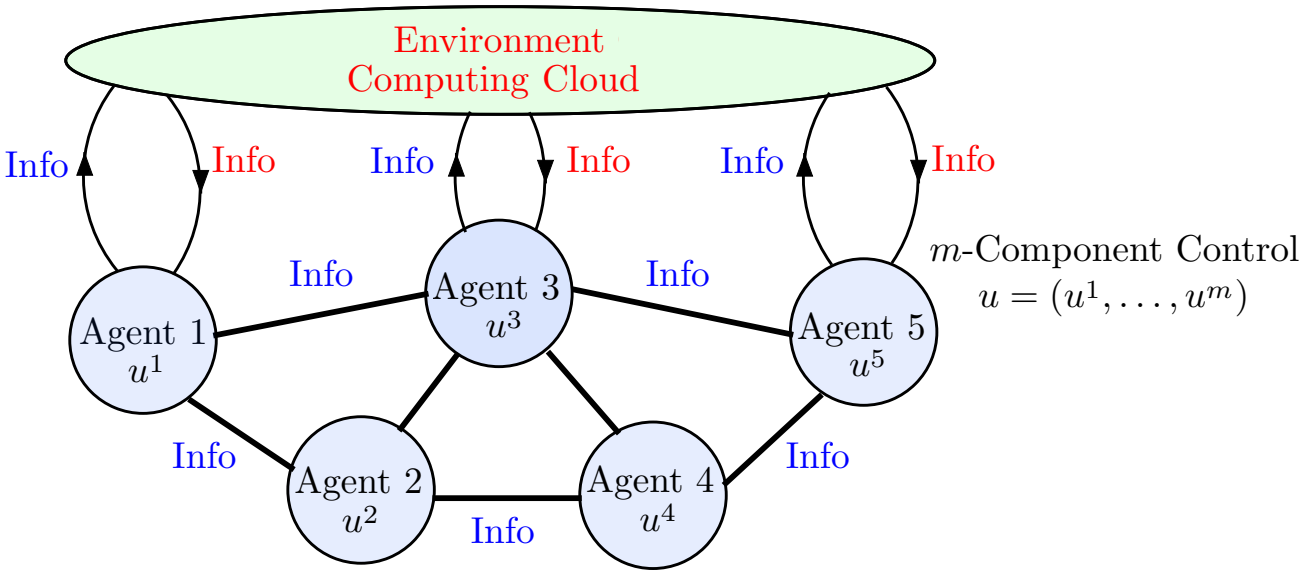
Lecture 7

Applications of Approximation in Value Space to Multiagent and Multiple Objects Tracking Problems

Outline

Outline

Multiagent Problems (1960s $\rightarrow$)

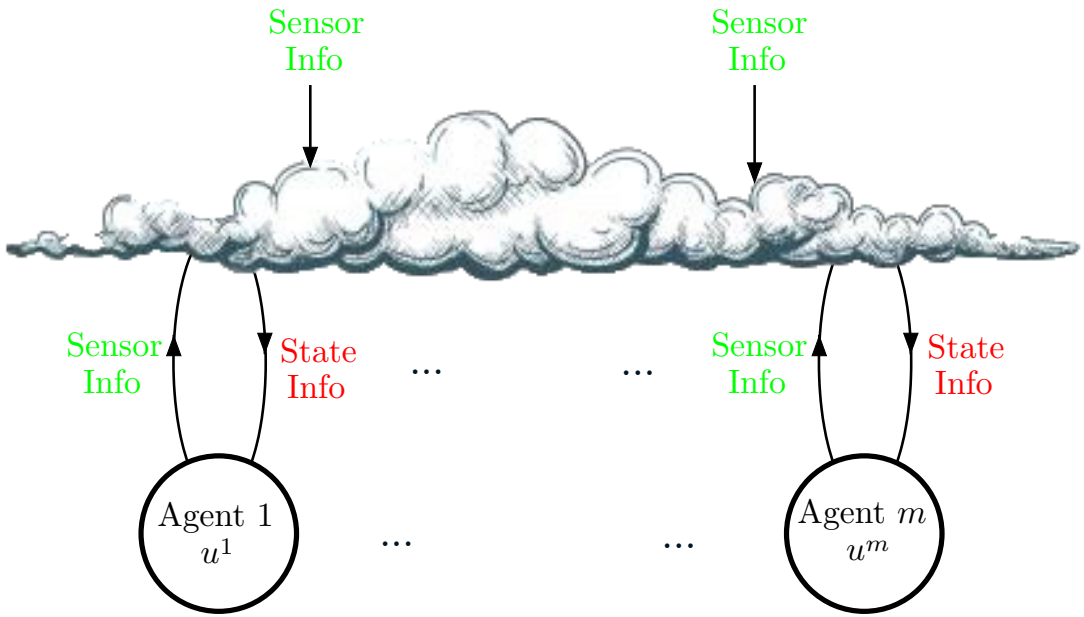


- Multiple agents collecting and sharing information selectively with each other and with an environment/computing cloud
- Agent i applies decision u^i sequentially in discrete time based on info received

The major mathematical distinction between problem structures

- The classical information pattern: Agents are fully cooperative, fully sharing, and never forgetting information. Can be treated by DP
- The nonclassical information pattern: Agents are partially sharing information and may be antagonistic. HARD because it is hard to treat by DP

Starting Point: A Classical Information Pattern

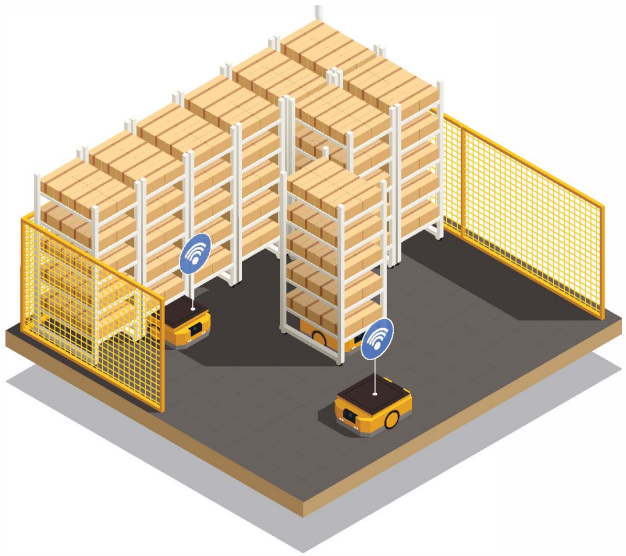


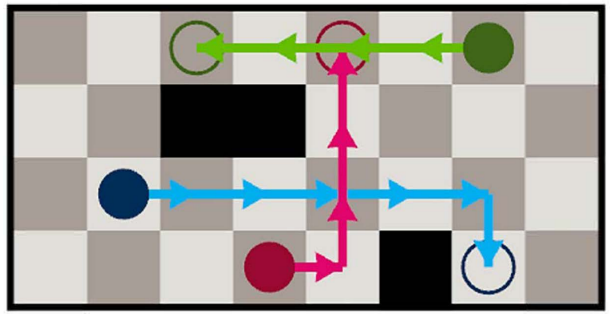
At each time: Agents have exact state info; choose their controls as a function of state

Model: A discrete-time (possibly stochastic) system with state x and control u

- Decision/control has m components $u = (u^1, \ldots, u^m)$ corresponding to m "agents"
- "Agents" is just a metaphor - the important math structure is $u = (u^1, \ldots, u^m)$
- The theoretical framework is DP. We will reformulate for faster computation
  - We first aim to deal with the exponential size of the search/control space
  - Later we will discuss how to compute the agent controls in distributed fashion (in the process we will deal in part with nonclassical info pattern issues)

Multiagent Path Finding Example: Modeling

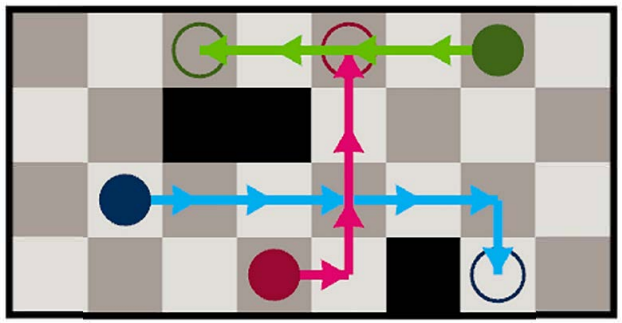




warehouse robots path planning $\Longrightarrow$ grid world representation

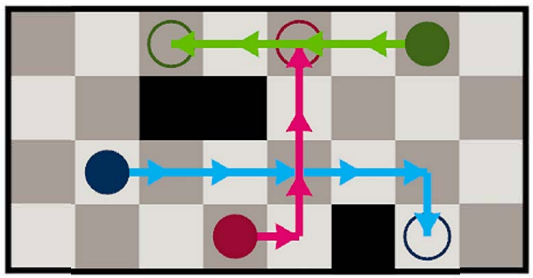
- There are $m = 3$ agents (solid circles) moving in 4 directions or standing still with perfect vision
- The agents have been assigned to some targets (open circles with the same color).
- The objective: reaching their respective targets in minimum time while avoiding collision with each other
- Simple heuristic: each agent follows the shortest path to the respective target, assuming other agents are not present (arrows in the figure)

Multiagent Path Finding Example: DP Formulation

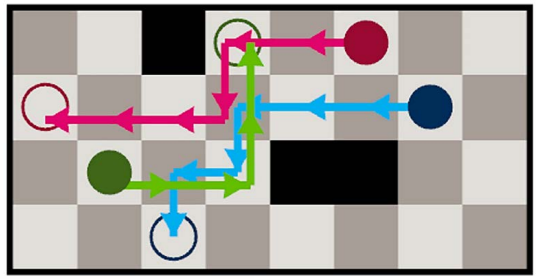


- States: current positions of all agents and their respective targets
- Control: each agent has at most 5 choices, their combination grows exponentially with m
- Stage cost: related to the number of collisions and the number of reached targets

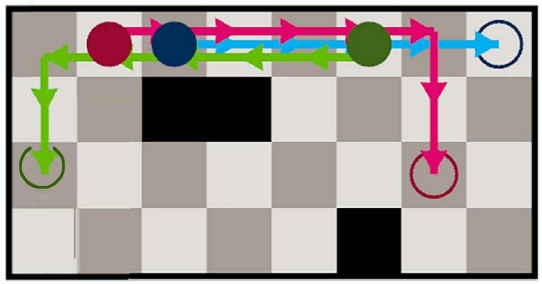
Multiagent Rollout for Multiagent Path Finding



case 1



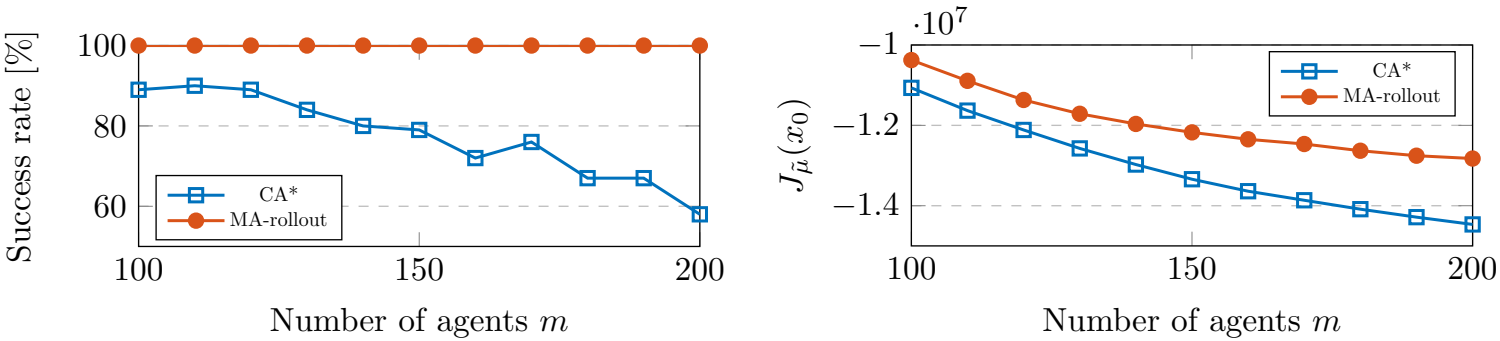
case 2



case 3

- At each time we must select one out of $\approx 5^m$ joint move choices
- Multiagent rollout reduces it to $5 \cdot m$ (while maintaining good properties)
- Key idea: Break down the control into a sequence of one-agent-at-a-time moves
- Each stage involves the following sequence of operations:
  - Minimizing Q-factors associated with the first agent, while the remaining two agents follow base heuristics
  - Minimizing Q-factors associated with the second agent, while the last agent follows base heuristics
  - Minimizing Q-factors associated with the last agent
- We allow a change of the order in which the agents are selecting their controls, at every stage

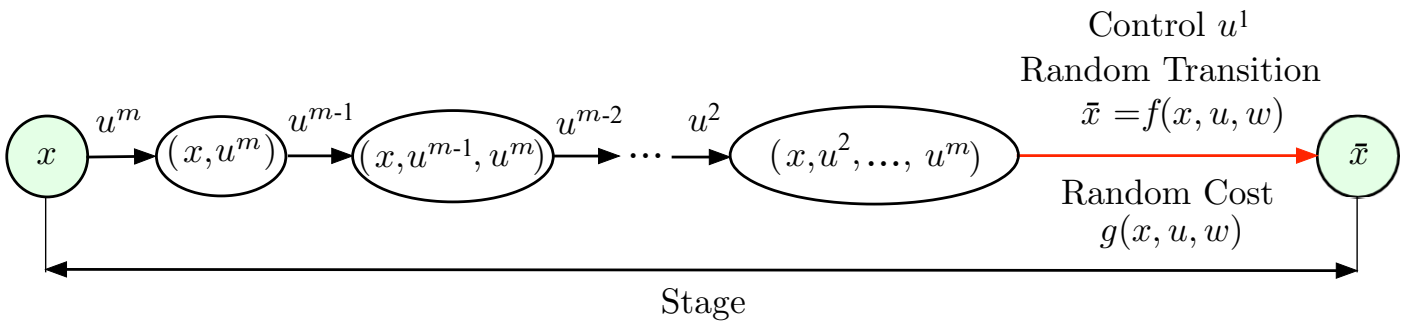
Test Results: Collision Avoidance at Comparable Cost



- The performance of the algorithm is compared with cooperative A* ('CA*' for short) introduced in [Sil05][a]
- 'CA*' calculates paths one-agent-at-a-time only at time 0
- In practice, multiagent rollout always finds feasible paths at comparable cost
- Scales well, up to $m = 200$ agents, with average computational time around 50 ms. Can also adapt to a changing environment through path replanning. See paper [ERL23][b] as well as implementation in C++[c]

[a] [Sil05] Silver, Cooperative Pathfinding
[b] [ERL23] Emanuelsson et. al., Multiagent Rollout with Reshuffling for Warehouse Robots Path Planning
[c] `https://github.com/will-em/multi-agent-rollout`

Implementation Variants of Multiagent Rollout



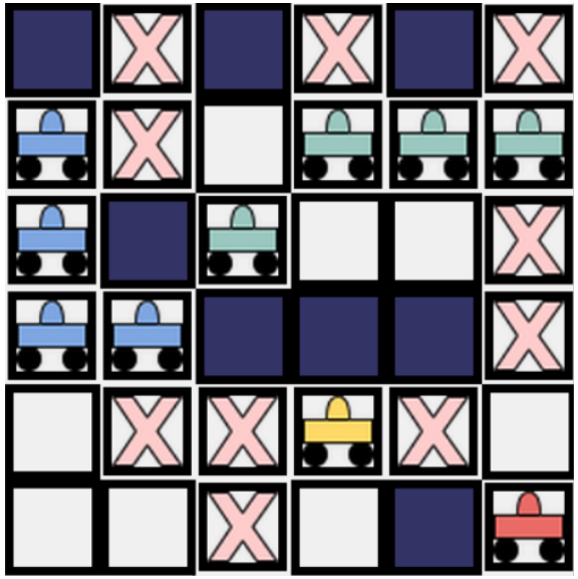
- Reshuffling the order of agents results in a different, yet still equivalent problem
- Multiagent rollout allows parallel computation of Q-factors
- Multiple base heuristics can be applied to enhance the performance further
- All those ideas are independent of each other and can be combined
- See textbook for additional material for order optimization and other variants

Outline

Section Outline

- Review of the multivehicle routing problem (MVRP).
- Generalization of MVRP for nonclassical information patterns.
- Overview of a distributed computing approach.
- The Decentralized Multiagent Rollout Algorithm (DMAR).
- Experimental Study: The impact of communication and coordination on performance.
- Physical implementations.
- Conclusions, extensions, and paper access.

Review: Multivehicle Routing Problem (MVRP)



Problem Definition

- **Instance:**
  - Undirected rectangular grid discretizing 2D space.
  - Set of locations representing *obstacles*.
  - Set of *agents*.
  - Set of *task* locations.
- Instance is *solved* when each task is visited at least once by some agent.
- **Objective:** Compute a set of trajectories that solve the instance such that the total number of vehicle movements is *minimized*.
- Can be treated directly with DP when *classical information pattern* is assumed.

The Unmapped Vehicle Routing Problem with Local Constraints (UMVRP-L)

Generalization

- Instance is not known a priori. There is no centralized computational cloud.
- Inter-agent communication and sensing is restricted to a *sensing radius* of k hops on the grid.
- *Nonclassical information pattern*.
- MVRP is a special case of UMVRP-L with $k = \infty$.
- UMVRP-L not yet been considered in the literature until now!

Approach

- We can treat the problem with DP-based decomposition.
- **Distributed Computing Approach:** Agents are processors with persistent memory and are capable of limited sensing, locomotion, and local communication.
- **Strategy**: Generate subinstances locally around agent clusters where information can be shared between agents (i.e. small MVRP instances), then apply approximate DP to the subinstances individually. Use randomized exploration to search for distant tasks.

Applications

Minefield disarmament, post-disaster search and rescue, resource discovery/exploitation.

Distributed Model

- **Configuration:** Processors w/ internal states arranged in a *communication network*.
- Communication between processors is facilitated via *message passing* through ports.
- **Events:** Computation and message send/receive updates a processor's state.
- **Distributed Algorithms**: $\text{Config}_0 \rightarrow \text{Event}_1 \rightarrow \text{Config}_1 \rightarrow \text{Event}_2 \rightarrow \text{Config}_2 \ldots$

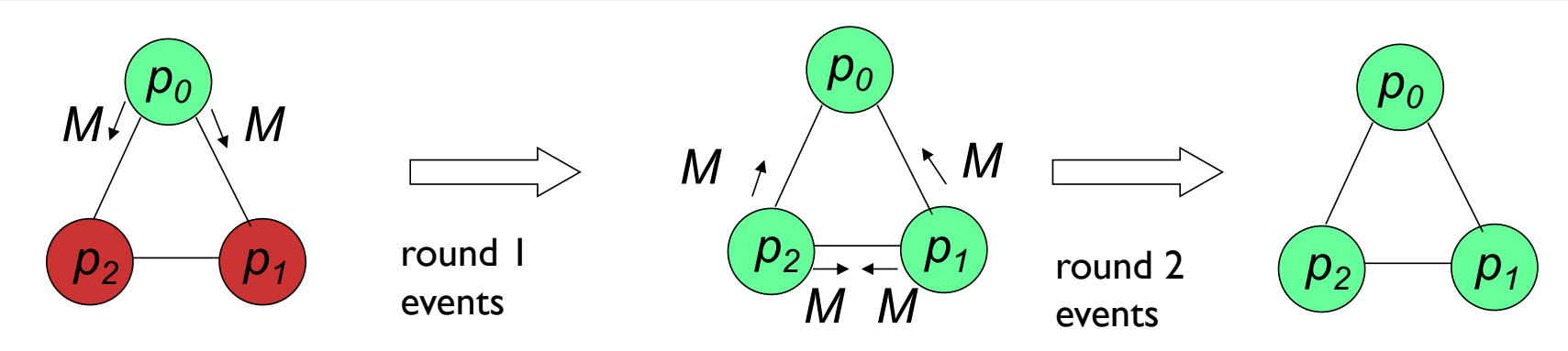
Algorithm Example Distributed Algorithm for Processor p

---

if Message in buffer and $p.\text{STATE} = \text{RED}$ **then**
  Set $p.\text{STATE} \leftarrow \text{GREEN}$; Forward messages along all ports.

Synchronous Schedulers

Processors repeatedly "wake up" and execute their common algorithm simultaneously.



Decentralized Multiagent Rollout (DMAR)

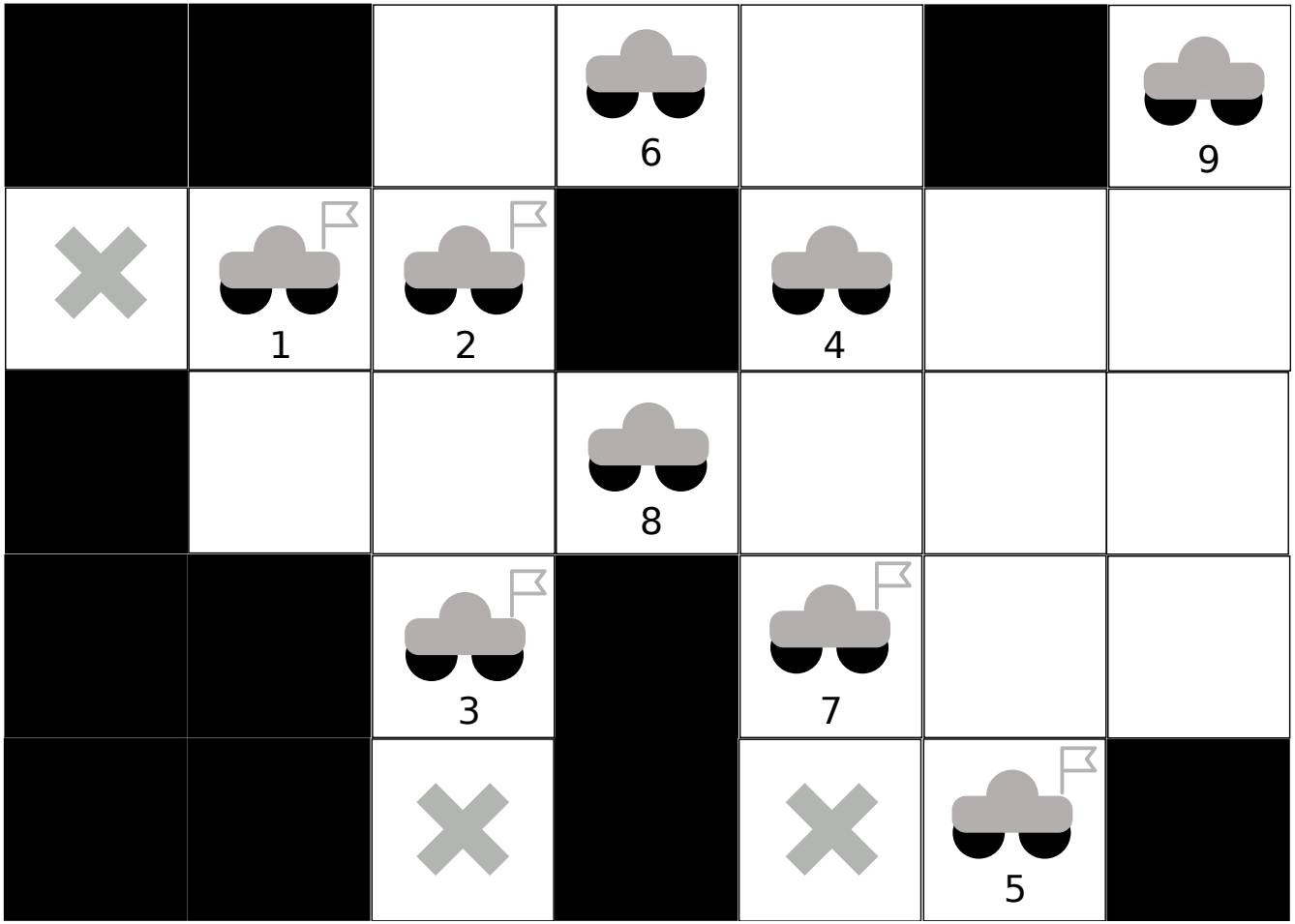
- DMAR is an iterative distributed algorithm that specifies state updates and message handling, leading to the collective behavior below.
- The *view* of an agent a includes all instance information available within k hops of a on the grid.

DMAR

Repeat:

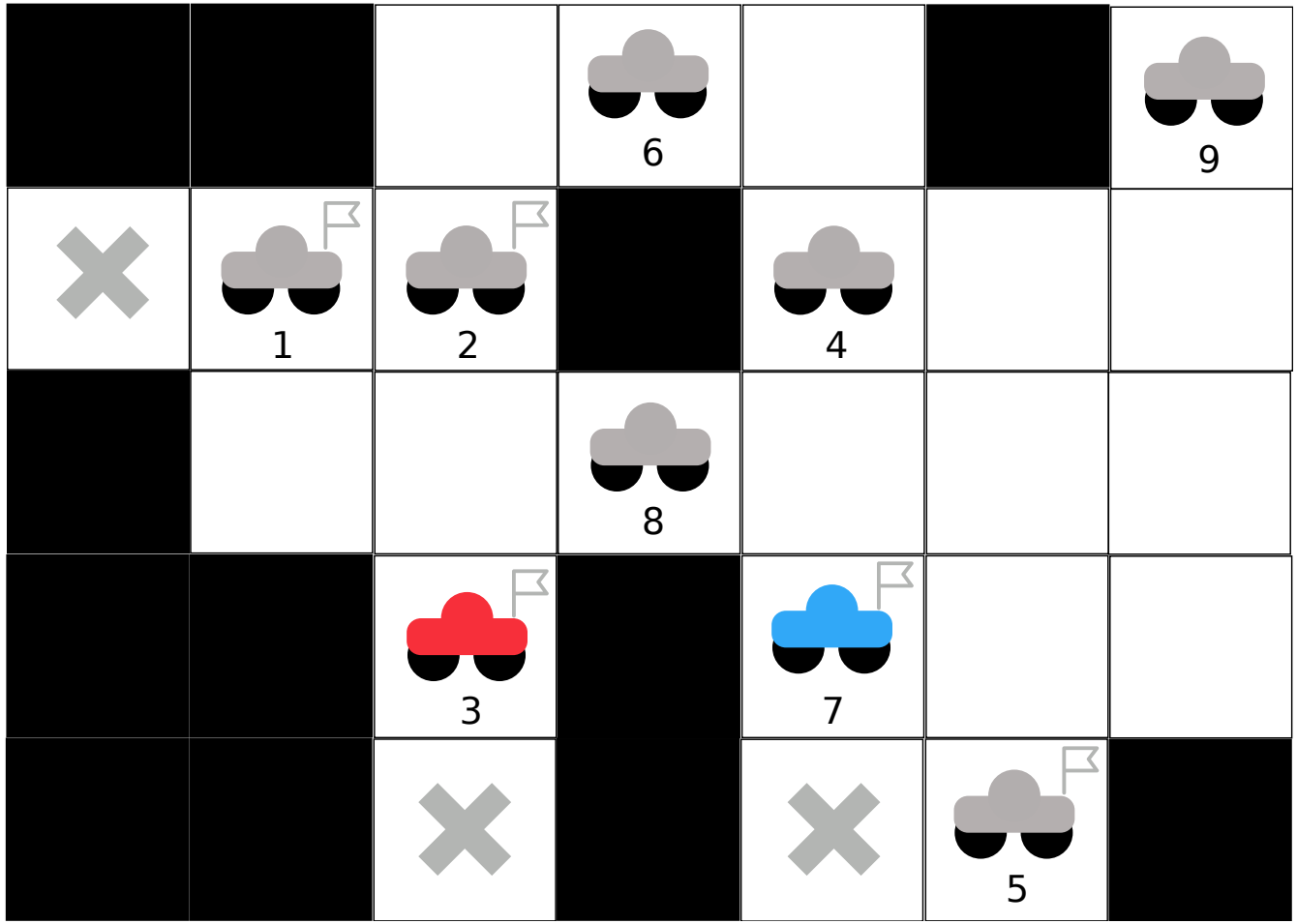
1. Agents self-organize locally into *clusters* of constant size around "leaders". Agents within clusters form a tree communication network, whose root is the leader. Tree structure facilitates flow of messages. Smaller adjacent clusters may be combined into larger ones.
2. For each cluster K, agents in K route their views to the cluster leader.
3. For each cluster K, the cluster leader computes a set R of control sequences according to multiagent rollout with greedy base policy using assembled view information.
4. Sequences R are broadcast through agent tree.
5. Each agent moves along its assigned control trajectory, then unassigns self from cluster.
6. Agents not in clusters generate and follow random trajectories of bounded length, searching for tasks.

Decentralized Multiagent Rollout: Cluster Formation



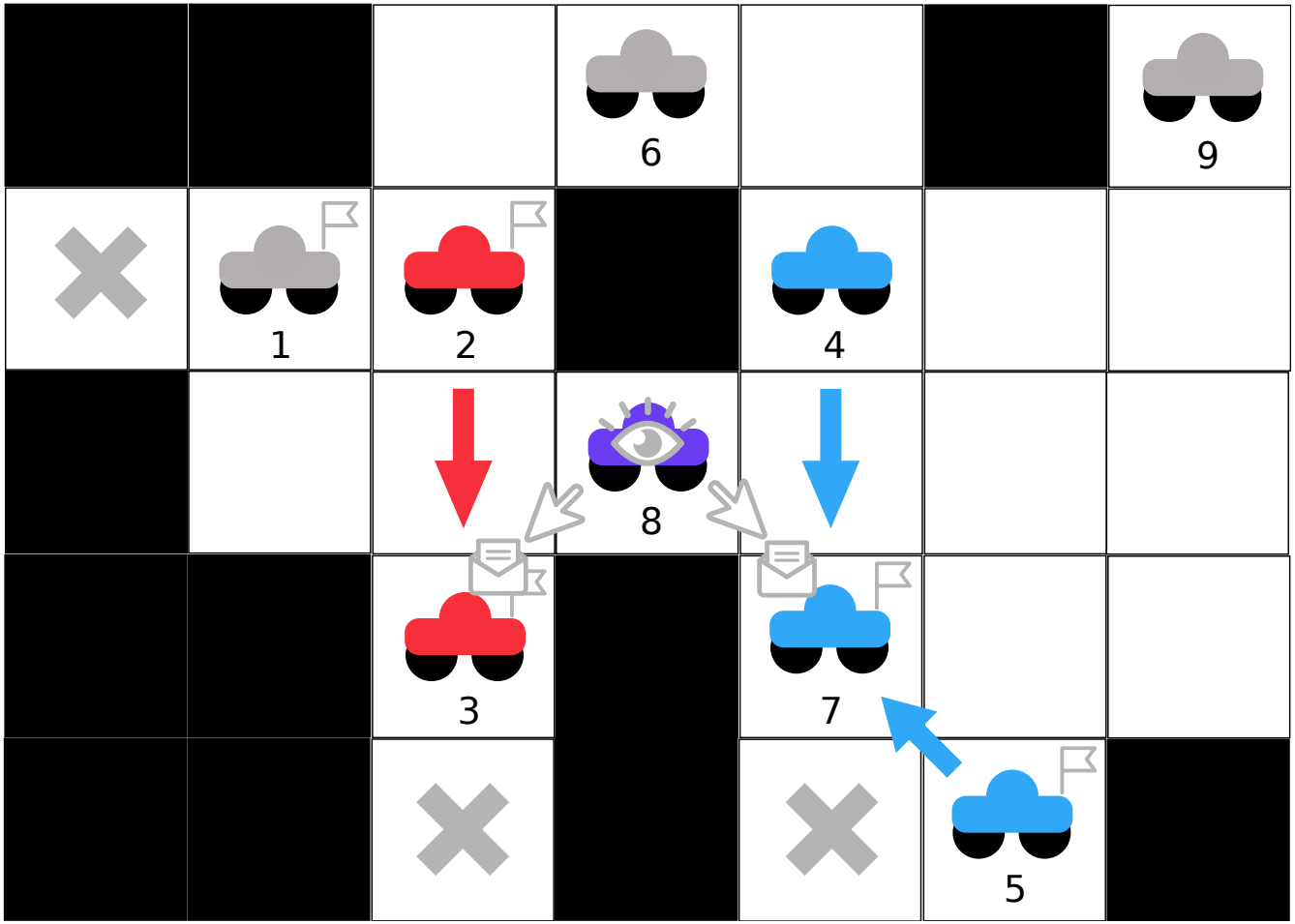
- A function on a size parameter bounds the size of the largest cluster.
- Let $k = 2$ (recall number of hops).
- Color indicates cluster membership; gray is default.

Decentralized Multiagent Rollout: Cluster Formation



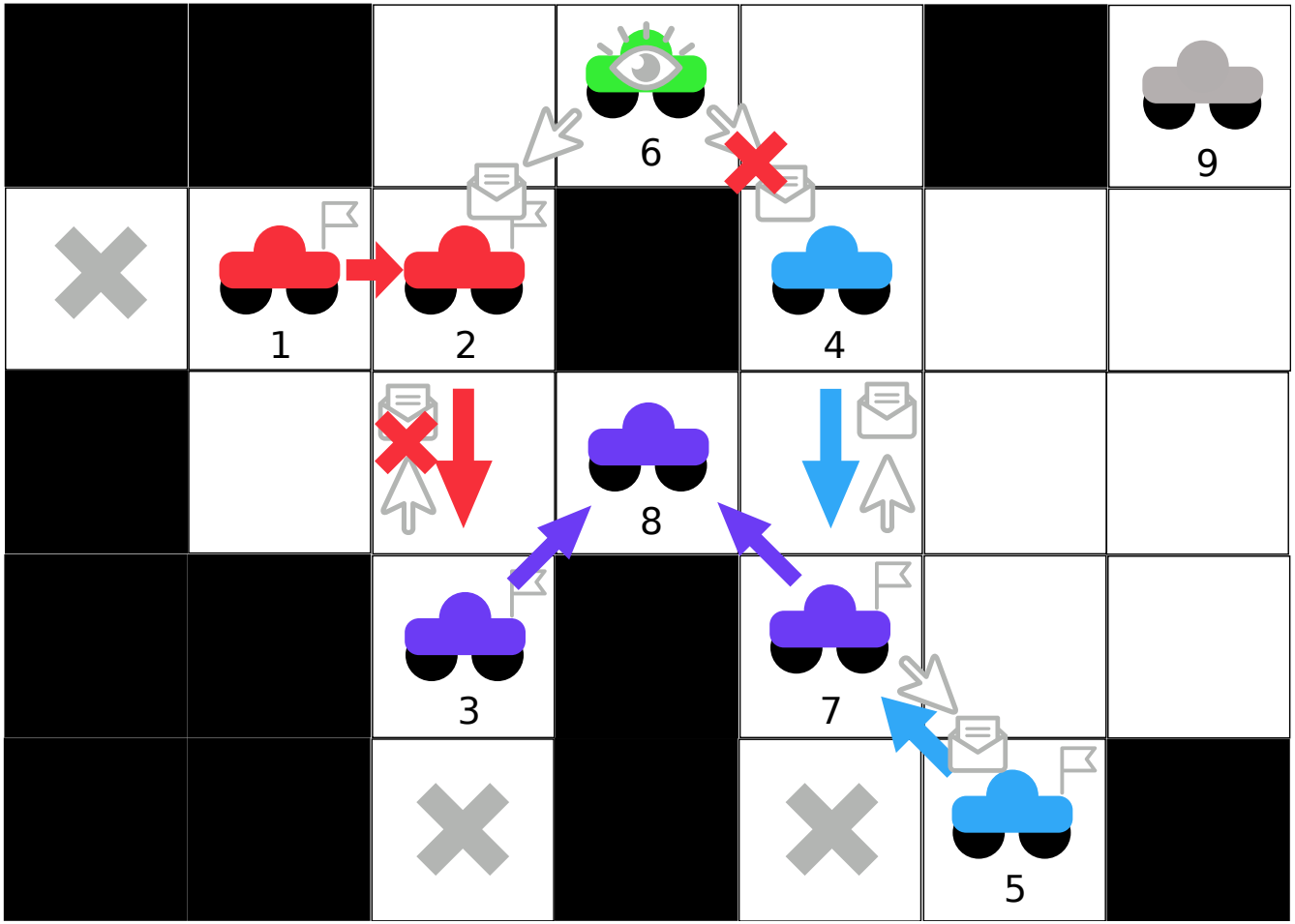
- A function on a size parameter bounds the size of the largest cluster.
- Let $k = 2$ (recall number of hops).
- Color indicates cluster membership; gray is default.

Decentralized Multiagent Rollout: Cluster Formation



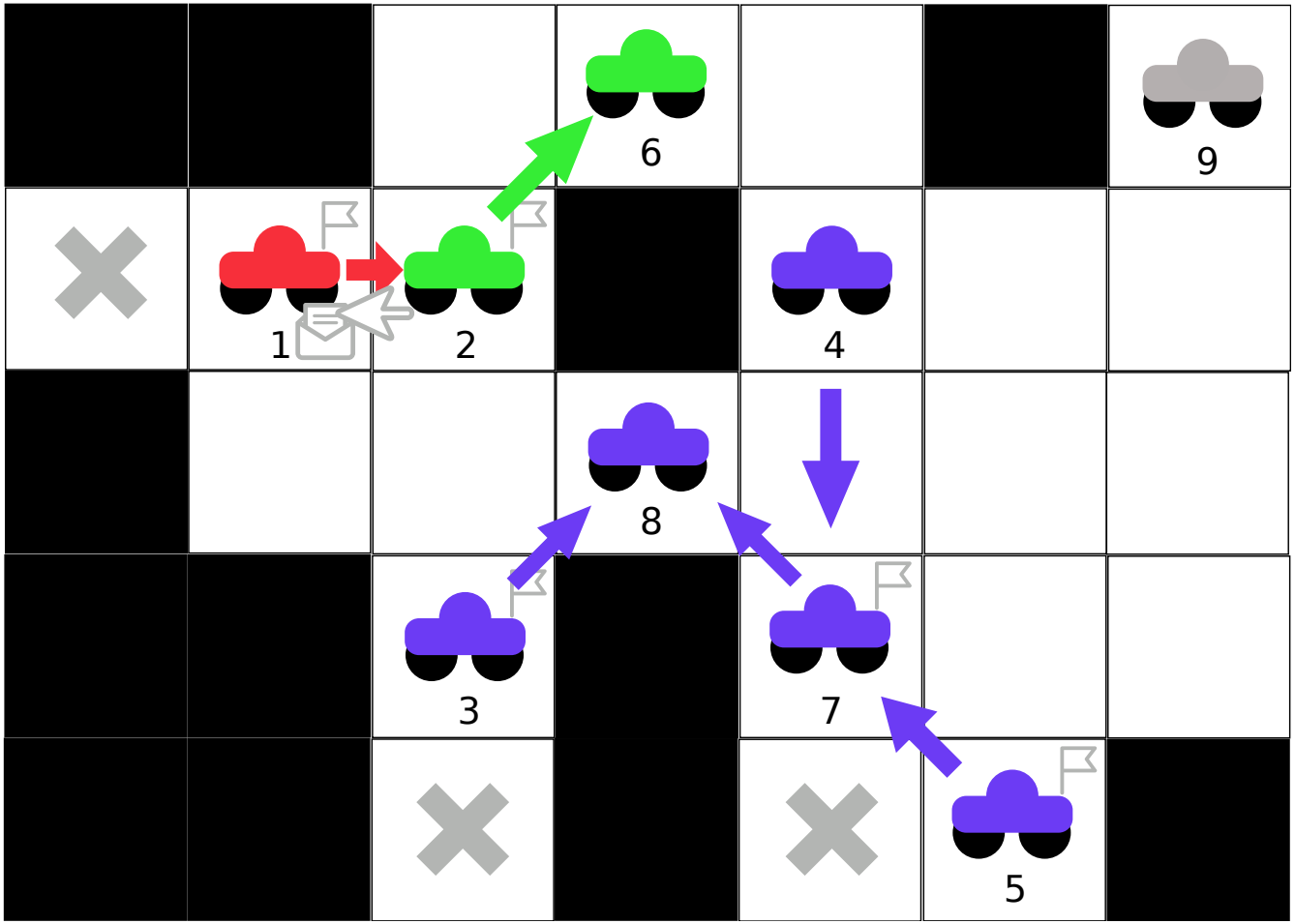
- A function on a size parameter bounds the size of the largest cluster.
- Let $k = 2$ (recall number of hops).
- Color indicates cluster membership; gray is default.

Decentralized Multiagent Rollout: Cluster Formation



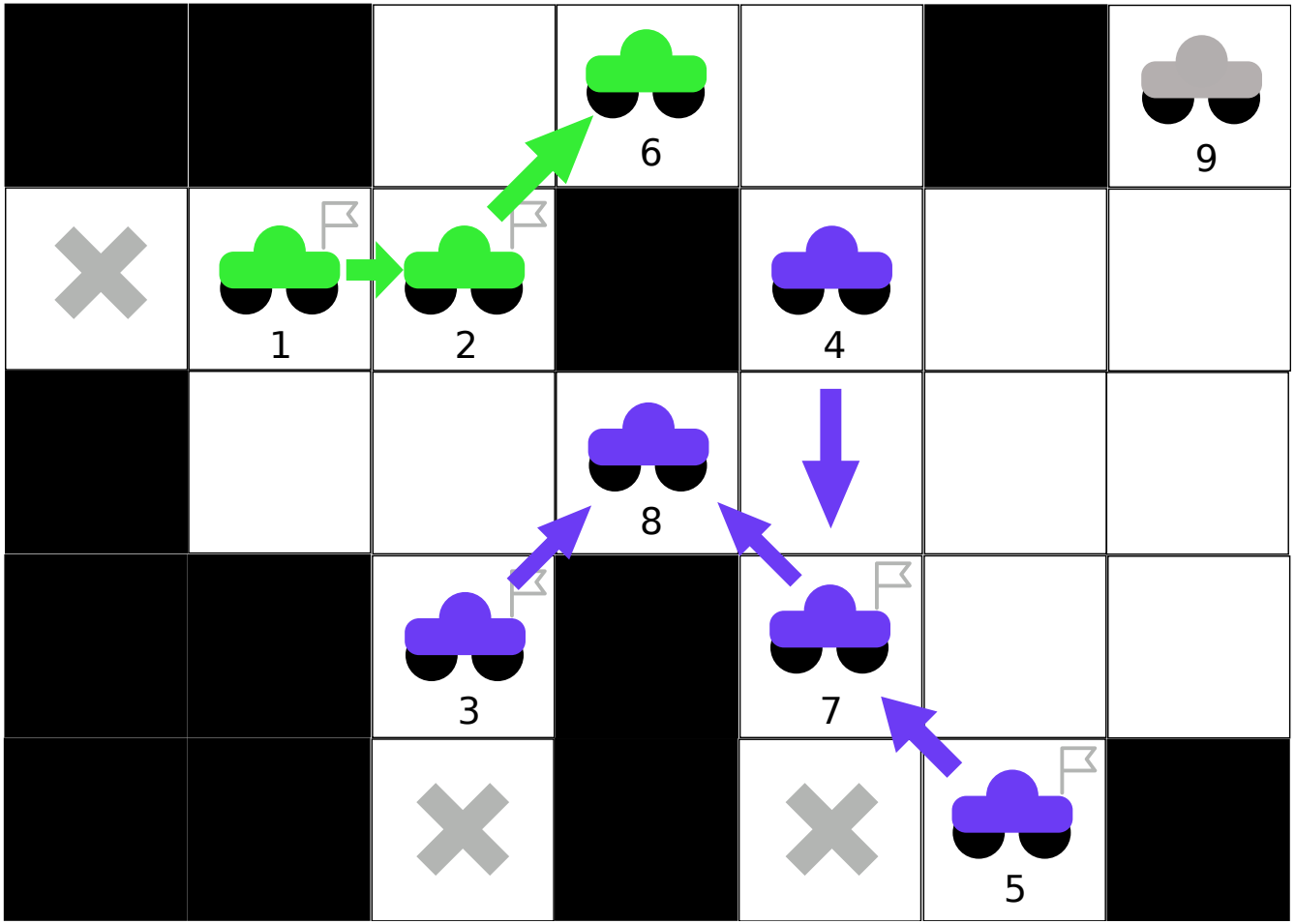
- A function on a size parameter bounds the size of the largest cluster.
- Let $k = 2$ (recall number of hops).
- Color indicates cluster membership; gray is default.

Decentralized Multiagent Rollout: Cluster Formation



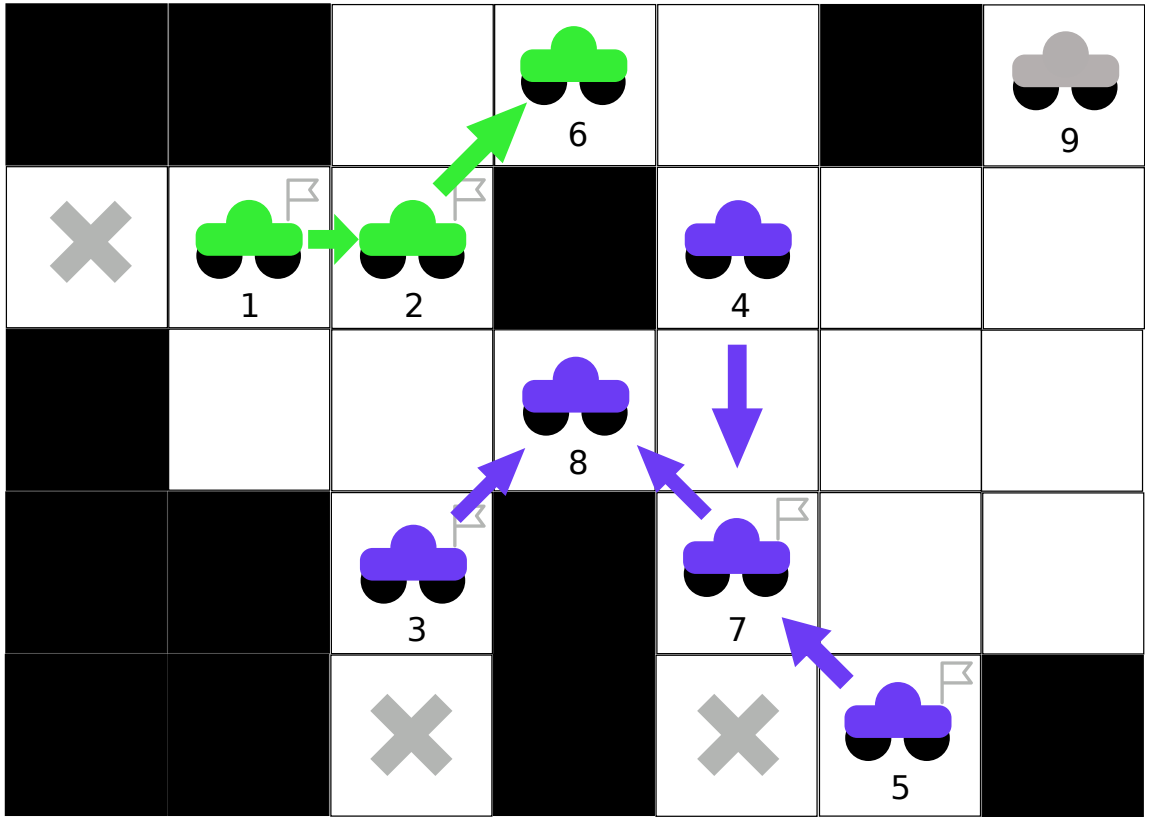
- A function on a size parameter bounds the size of the largest cluster.
- Let $k = 2$ (recall number of hops).
- Color indicates cluster membership; gray is default.

Decentralized Multiagent Rollout: Cluster Formation



- A function on a size parameter bounds the size of the largest cluster.
- Let $k = 2$ (recall number of hops).
- Color indicates cluster membership; gray is default.

Decentralized Multiagent Rollout: Everything Else



- Agents in clusters pass view information through tree edges until leader has full cluster map.
- Leader applies multiagent rollout to cluster map and broadcasts computed trajectories through the tree.
- Agents follow their received trajectories; Agents not in a cluster perform random walk of bounded length.

Observations about DMAR

Connections to Approximate DP

- When $k = \infty$, a single cluster is formed and rollout is performed with respect to the entire instance by a single processor.
- Reducing k approximates the rollout structure by decomposing the instance.
- The aggregation of these smaller solutions yields a solution to the full instance that approximates the rollout solution.
- Even when clusters are small, using multiagent rollout is imperative due to the curse of dimensionality.

Notes on Distributed Computing

- Distribued algorithms can be challenging to design.
- They rely on primitives such as leader election and message passing. Sometimes communication networks are dynamic as is the case here.
- Proving that a distributed algorithm produces a specific collective behavior is also challenging (deadlocks and invalid configurations).
- Randomized elements also require careful analysis.

Experimental Study: The impact of communication and coordination on performance.

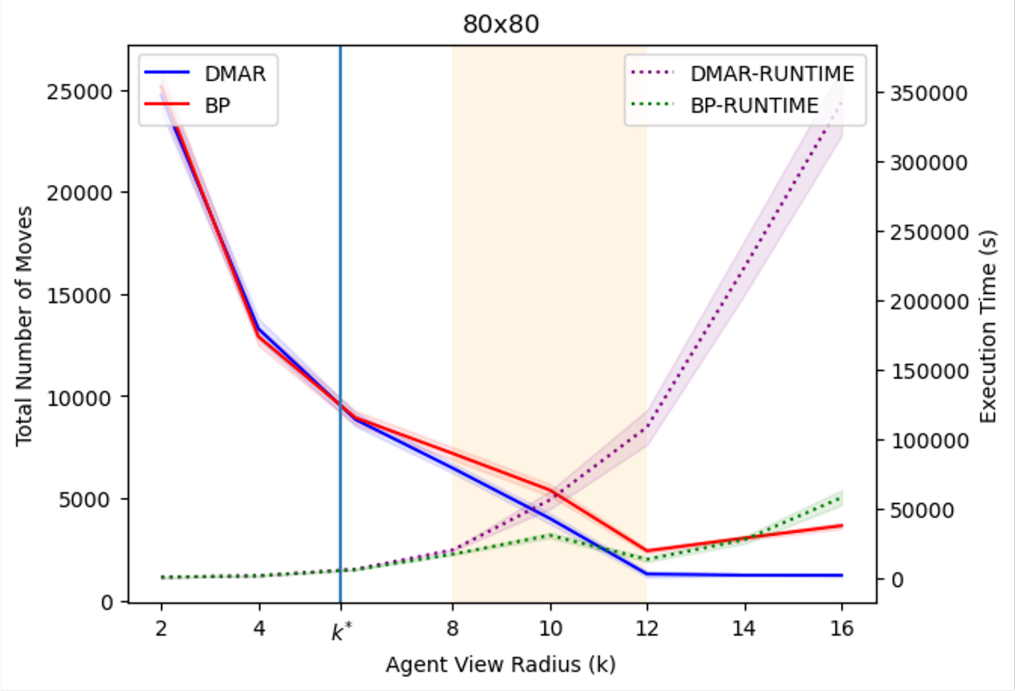
Goal of the Study

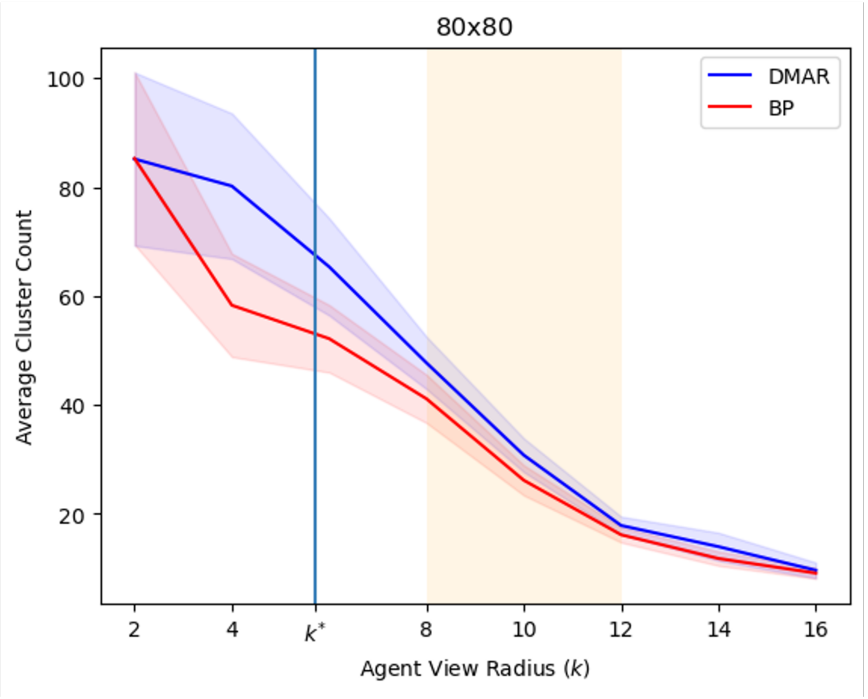
- We want to quantify the amount of local information necessary for DMAR to produce "good" solutions.
- As there are no benchmarks for UMVRP-L, we consider a greedy strategy (we call this the *base policy*) that involves no coordination that can be applied to UMVRP-L.
- **Role of Coordination and Communication:** How large must the sensing radius k be such that DMAR outperforms the base policy?

Experimental Design

- We consider 8 grid classes: 10x10, 20x20,. . . ,80x80.
- For each class we consider three agent-to-task ratios (1:1,2:1,1:2).
- For each size/ratio combination we uniformly distribute obstacles over 20% of nodes and repeat 10 times.
- For each instance generated we distribute agents uniformly and execute DMAR ten times for each of several fixed k values.
- For each run we recorded total vehicle movements and wall-clock running times.
- The total number of simulations for the primary study is greater than 50,000.

Experimental Results: A Representative Sample

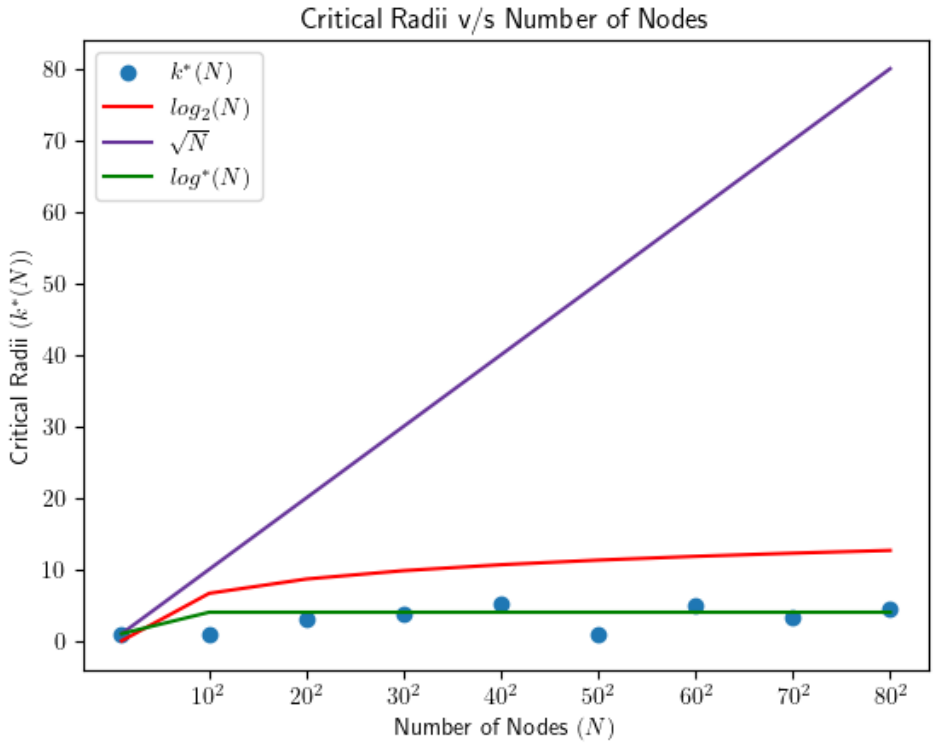




Observations

- Performance of DMAR degrades gracefully as k decreases.
- **Critical Radius** k^* where DMAR outperforms base policy.
- **Effective Range** just beyond k^* - we see a 2 to 4 factor increase in number of moves versus "centralized MAR" (larger radii).
- There is still a large number of clusters in effective range.

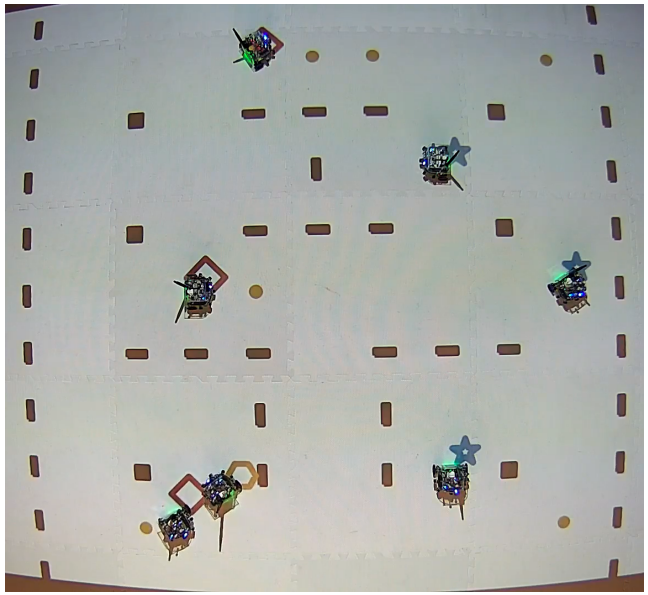
Aggregated Experimental Results



Observations

- Let N be the number of nodes in the network. Critical radii grow proportionally to $\log_2^*(N)$—a **very** slow growing function.
- Note that $\log_2^*(N) \leq 5$ for any $N \leq 10^{10000} \implies$ critical radius is effectively a small constant for any relevant instance.

Physical Implementations



Physical Experiments on the Robotarium Platform (at Georgia Tech)

- Planning by robots occurs in discrete space and time, but computed controls are then mapped to a continuous space velocity vector, and the new position is moved to with actuators.
- Simulations were repeated with sensing radius 20cm, 40cm, 60cm and 80cm on several instances.
- The results reflected our discrete simulations in terms of competing with the base policy.
- We have videos!

Conclusions of the Study

Conceptual Framework Coupling Classical and Nonclassical Information Patterns

- We explored the interface between classical and nonclassical information patterns by adapting methods used for the classical setting to the nonclassical setting.
- This led to the concept of clustering and applying multiagent decision making in a decentralized and parallel way.
- The critical quantity connecting classical vs nonclassical information patterns is the sensing radius.

Sensing Radius Conclusions

- Choosing a sensing radius in the effective range allows for greatly increased scalability at a small determiment to solution cost.
- One does not need to know the size of the network to choose a "good" sensing radius.
- The sensing radius must be large enough for good solutions, but not so large that agents may observe the entire network.

Applicability Conclusions

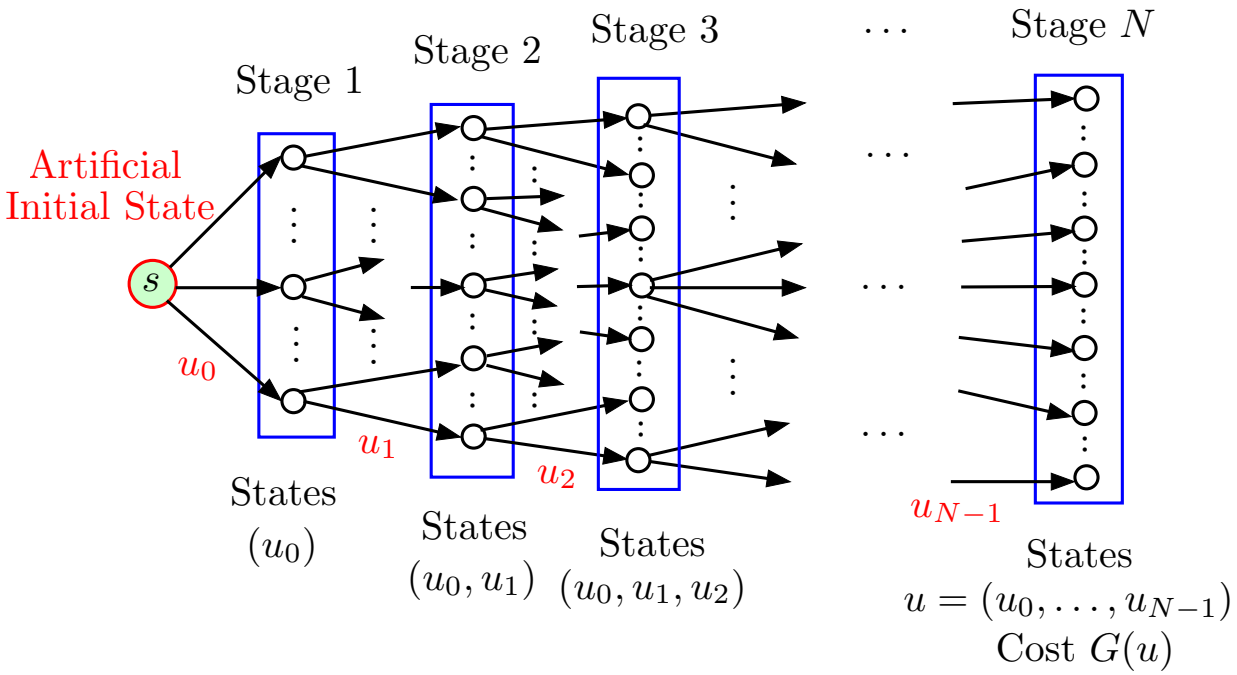
- Our approach is highly scalable and is conceptually adaptable to continuous space and robust to sensor noise, augmenting its real-world applicability.

Access the Paper

- Paper will appear in the proceedings of the 23rd International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2024). Link pending.
- arXiv link: `https://arxiv.org/abs/2305.15596`.

Outline

General Discrete Optimization



Minimize $G(u)$ subject to $u \in U$

- Assume that each solution u has N components: $u_0, \ldots, u_{N-1}$
- View the components as the controls of N stages
- Define $x_k = (u_0, \ldots, u_{k-1})$, $k = 1, \ldots, N$, and introduce artificial start state $x_0 = s$
- Define just terminal cost as $G(u)$; all other costs are 0

This formulation typically makes little sense for exact DP,
but often makes a lot of sense for approximate DP/approximation in value space

DP solution to the discrete optimization problem

- Start with
$$J_N^*(x_N) = G(x_N) = G(u_0, \ldots, u_{N-1}) \quad \text{for all } x_N \in U$$
- For $k = 0, \ldots, N-1$, let
$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} J_{k+1}^*(x_k, u_k) \quad \text{for all } x_k$$
where $U_k(x_k)$ need to be suitably defined.
- Construct the optimal solution $(u_0^*, \ldots, u_{N-1}^*)$ by forward calculation
$$u_k^* \in \arg \min_{u_k \in U_k(x_k)} J_{k+1}^*(x_k, u_k) \quad \text{for all } x_k$$

Approximation in value space

- Use some $\tilde{J}_{k+1}$ in place of J_{k+1}^*
- Starting from the artificial initial state, for $k = 0, \ldots, N-1$, set
$$\tilde{u}_k \in \arg \min_{u_k \in U_k(x_k)} \tilde{J}_{k+1}(x_k, u_k) \quad \text{for all } x_k$$

Multiple Object Tracking

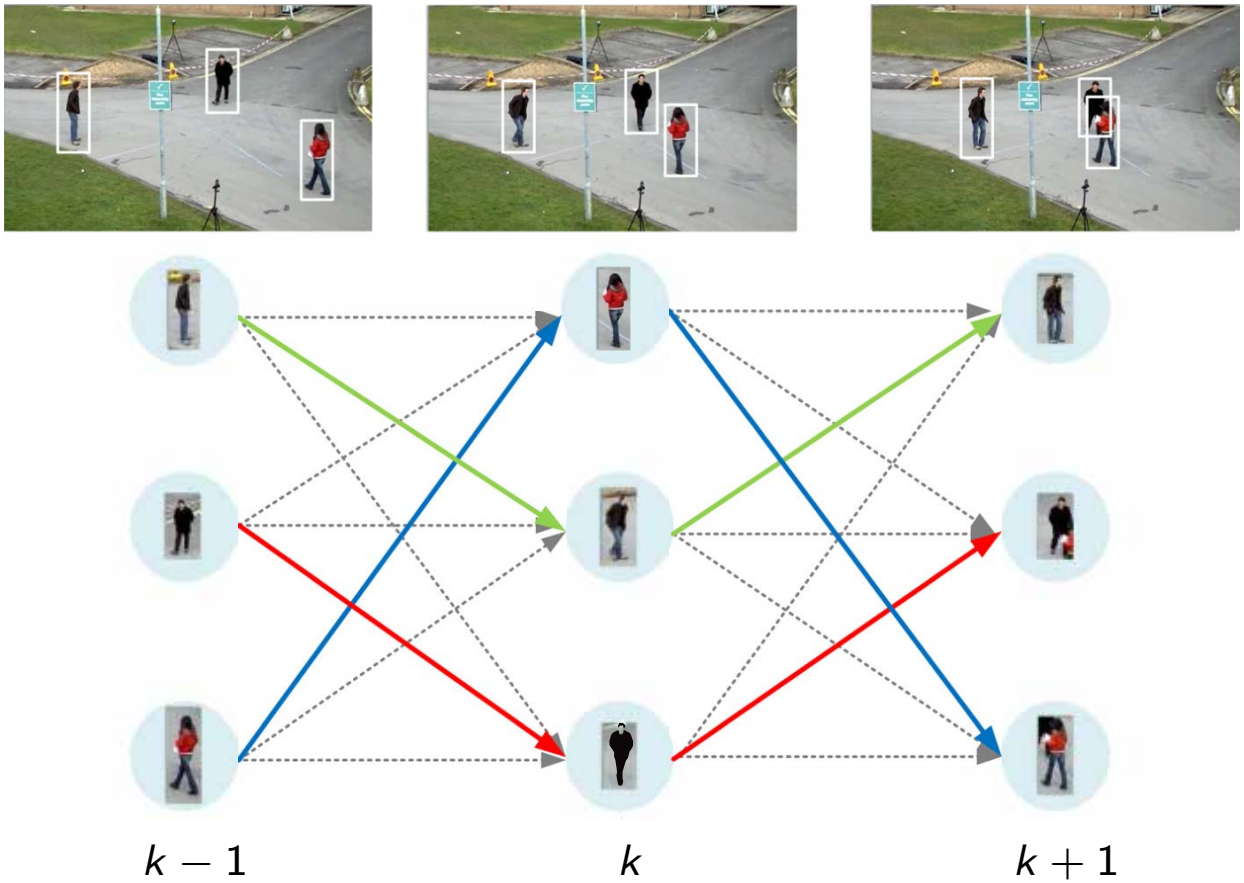
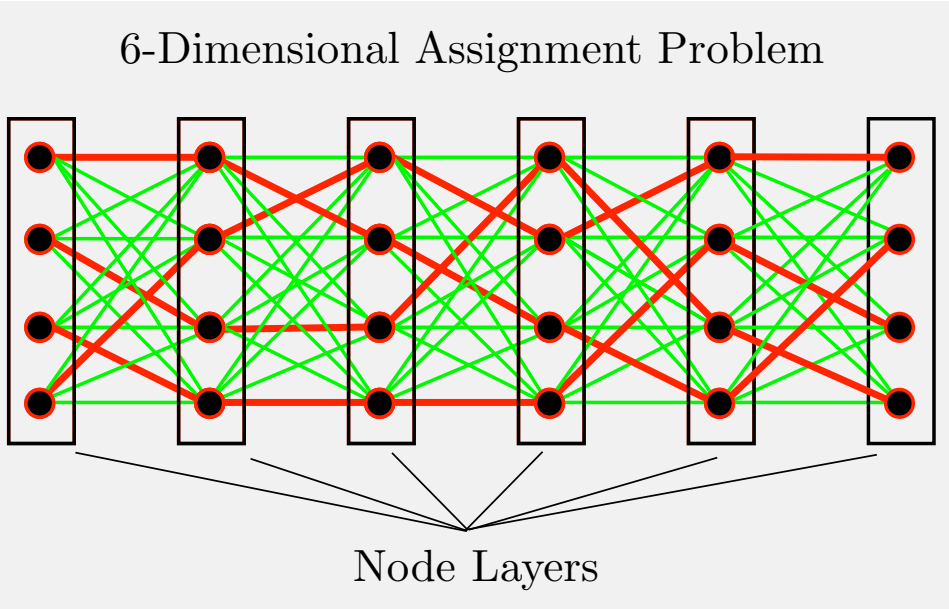


Figure source: [CGI22][1]

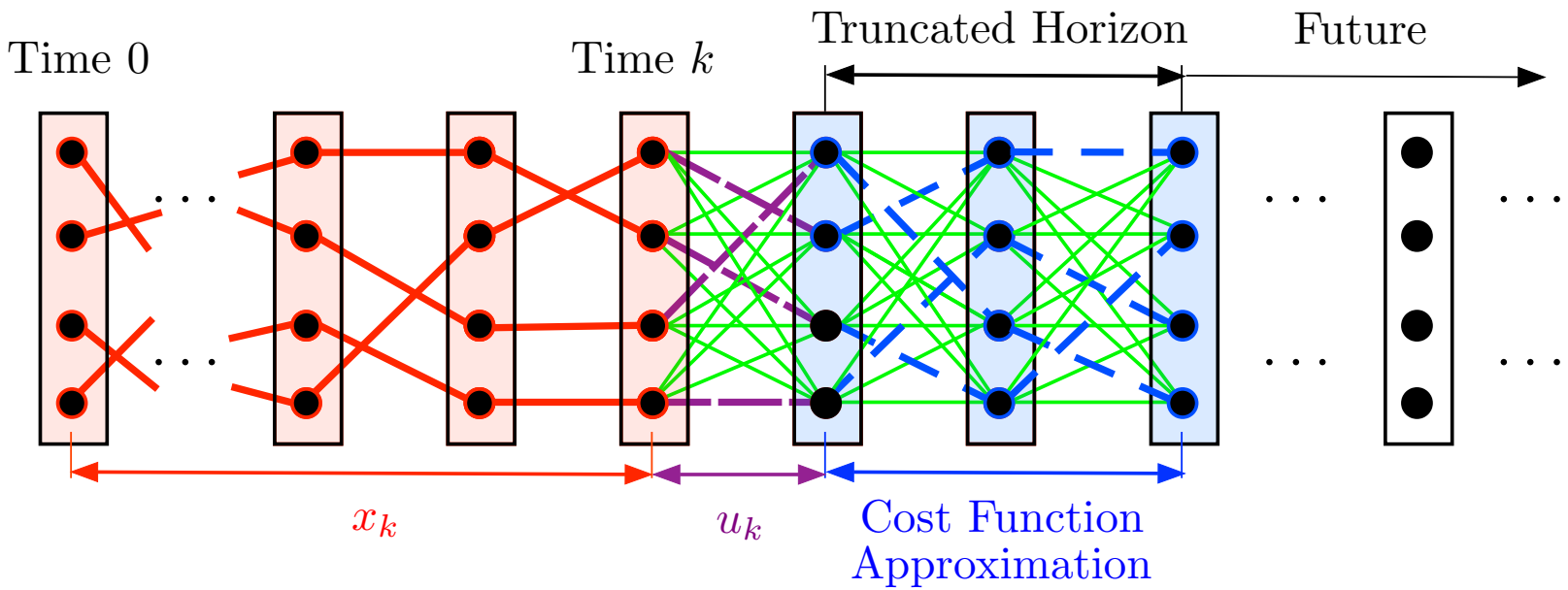
- Multiple object tracking (MOT) aims to match the same objects over various frames
- Nontrivial: occlusion, changes in object appearance, and real-time computation constraint
- Important problem with many applications: traffic monitoring, robotics, consumer analytics, augmented and virtual realities ...

Multidimensional Assignment Problem



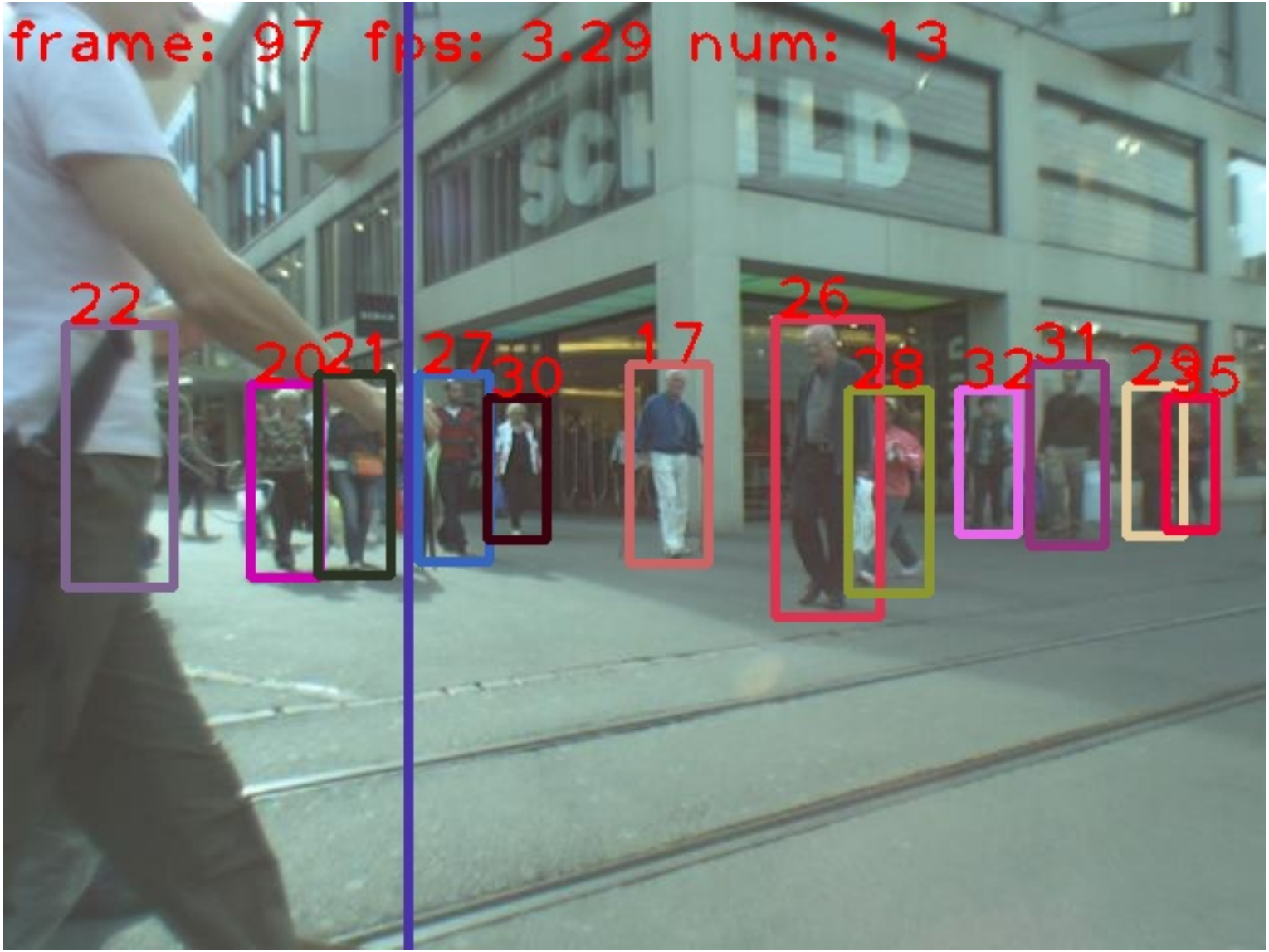
- MOT can be modeled as a multidimensional assignment problem
- There are $(N+1)$ layers (frames) of nodes
- A grouping consists of $N+1$ nodes $(i_0, \ldots, i_N)$ where i_k belongs to kth layer, and N corresponding arcs
- For each grouping, there is an associated cost depending on the entire grouping
- Our goal: find m groupings so that each node belongs to one and only one grouping and the sum of the costs of the groupings is minimized

Approximation in Value Space



- Approximation in value space involves the following key ideas:
  - One-step lookahead minimization
  - Truncated rollout
  - Cost approximation $\tilde{J}$ with structure that matches the assignment problem
- Q-factor minimization reduces to solving a 2-dimensional assignment problem
- Results: robust and consistent matching against occlusion

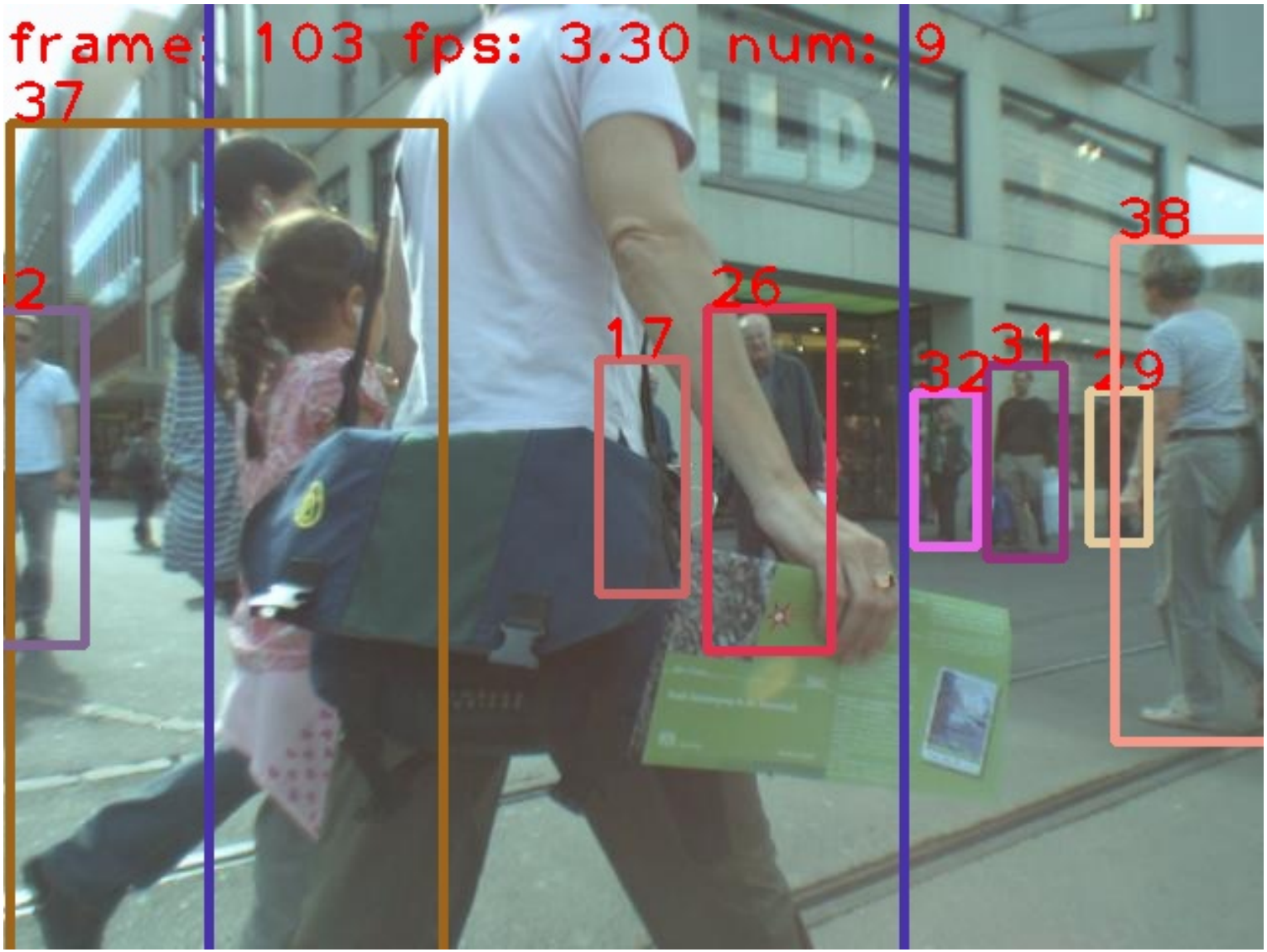
MOT Example: Base Heuristic



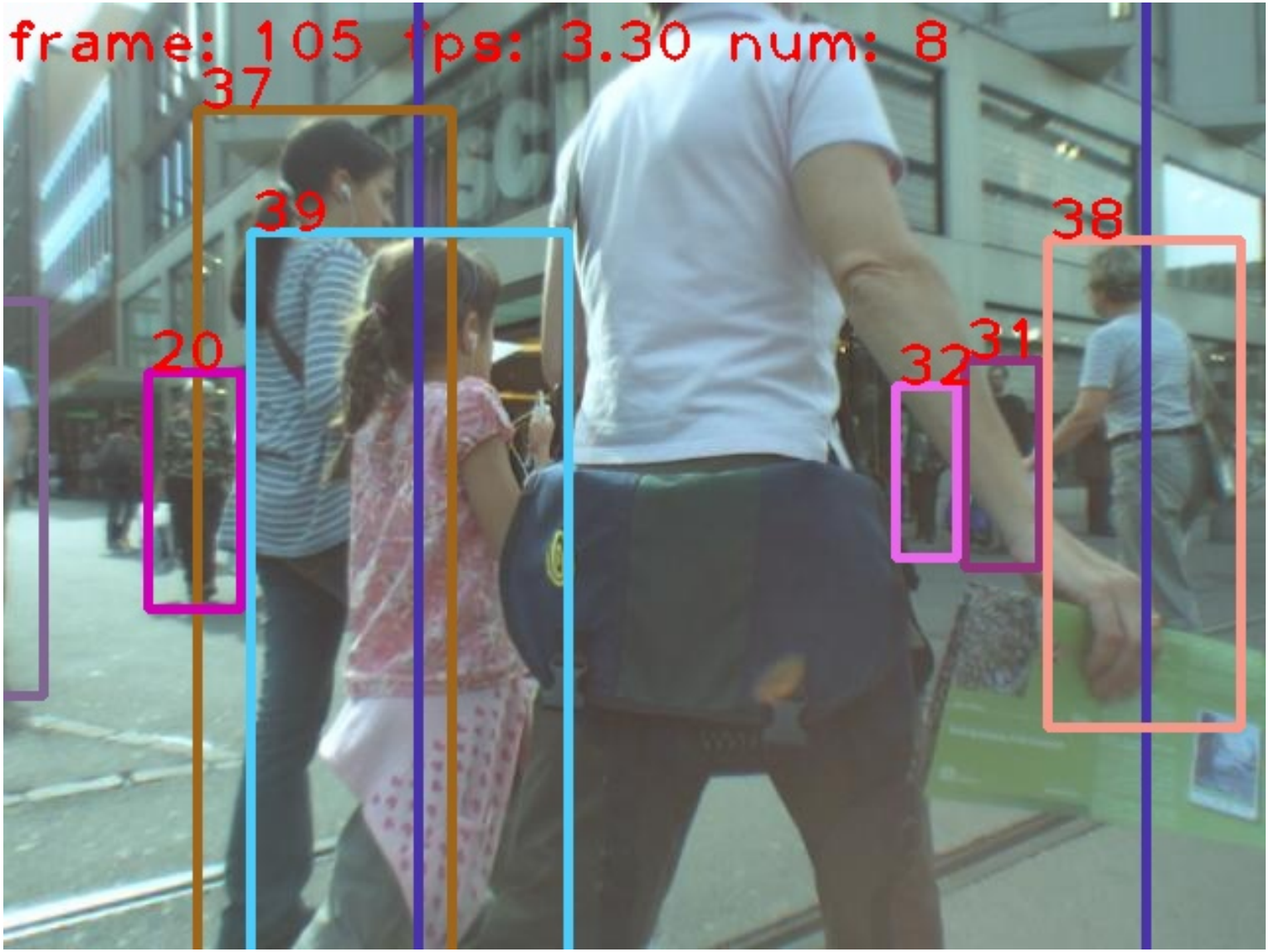
MOT Example: Base Heuristic

MOT Example: Base Heuristic

MOT Example: Base Heuristic



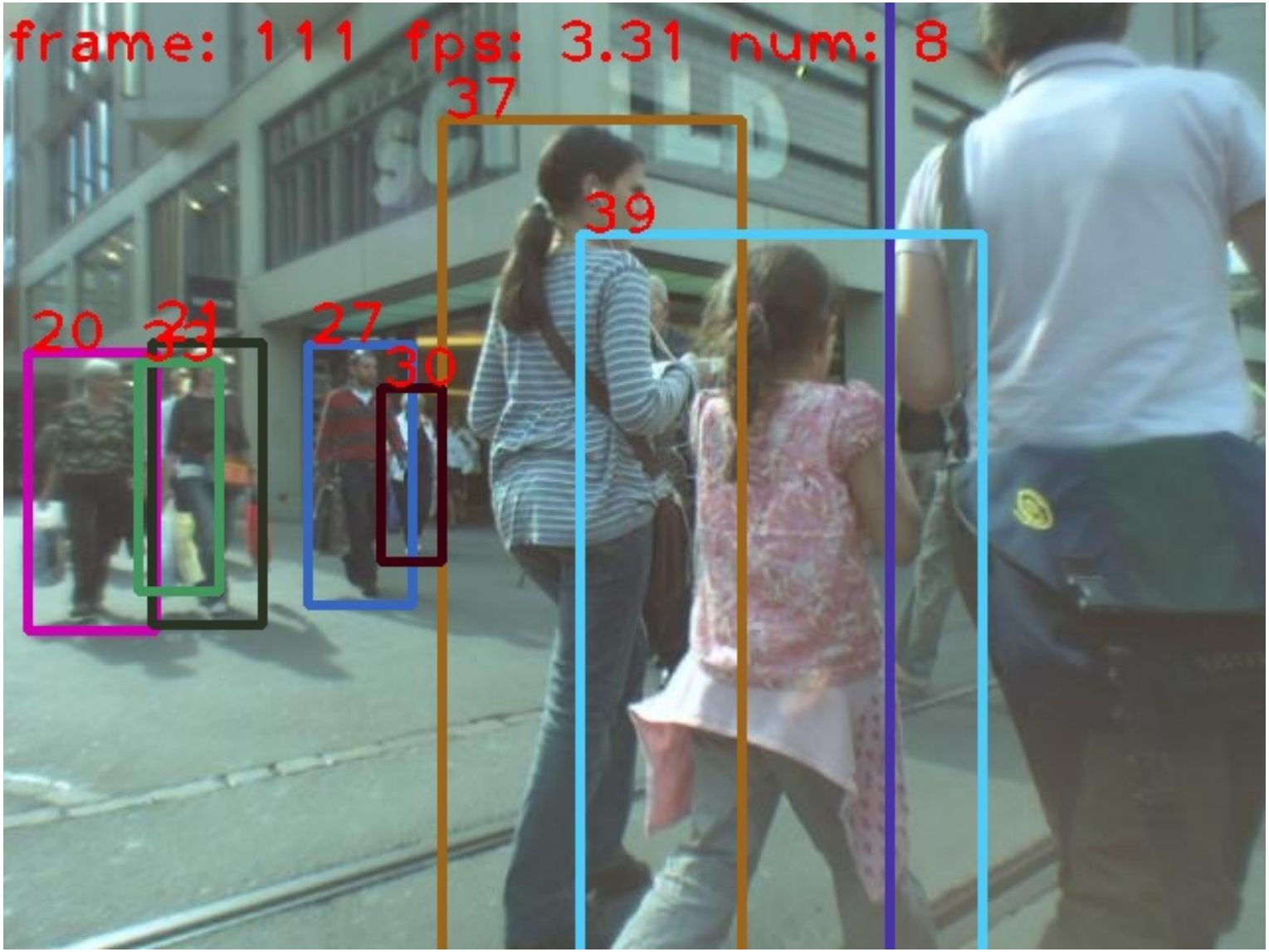
MOT Example: Base Heuristic



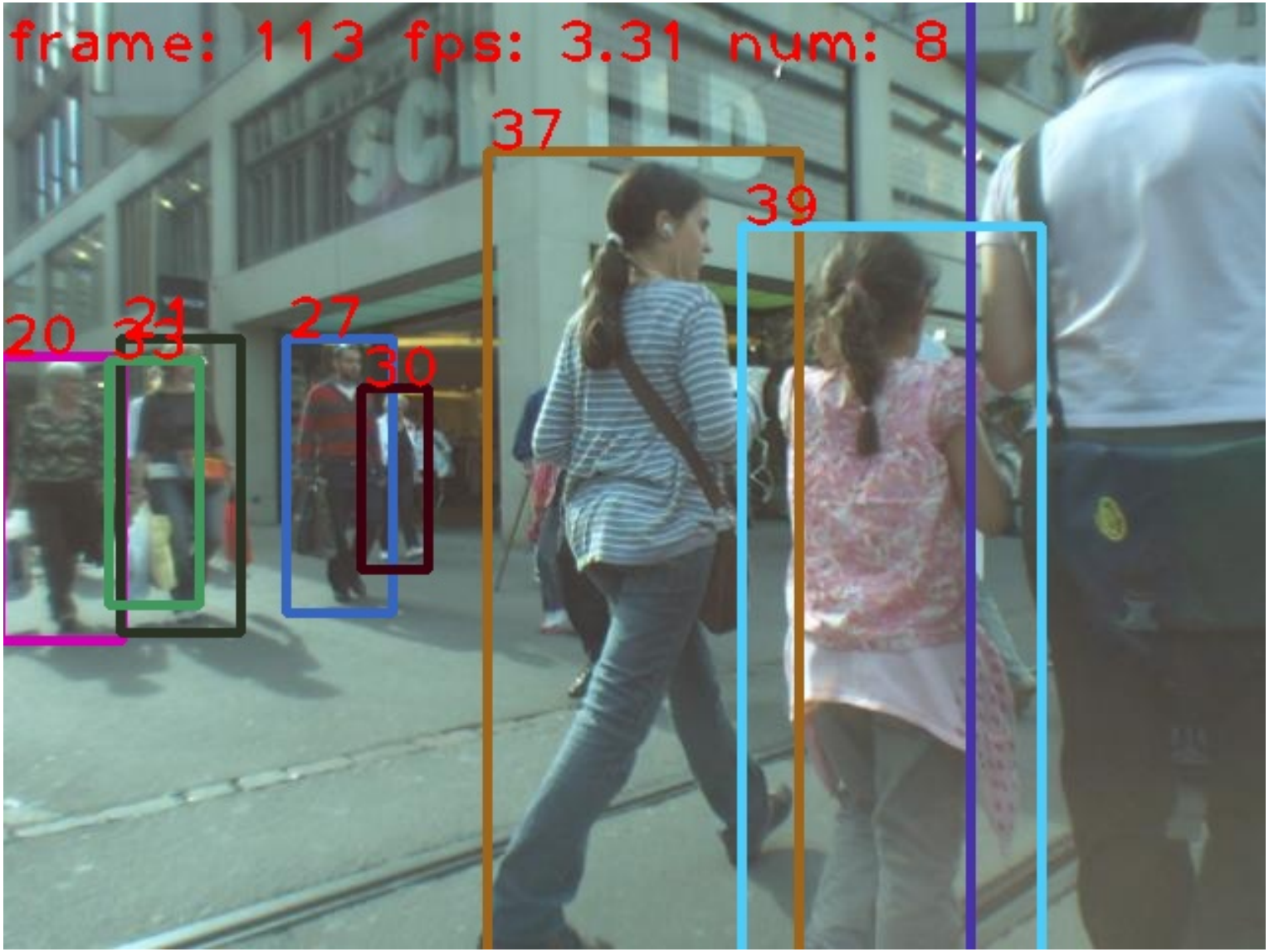
MOT Example: Base Heuristic

MOT Example: Base Heuristic

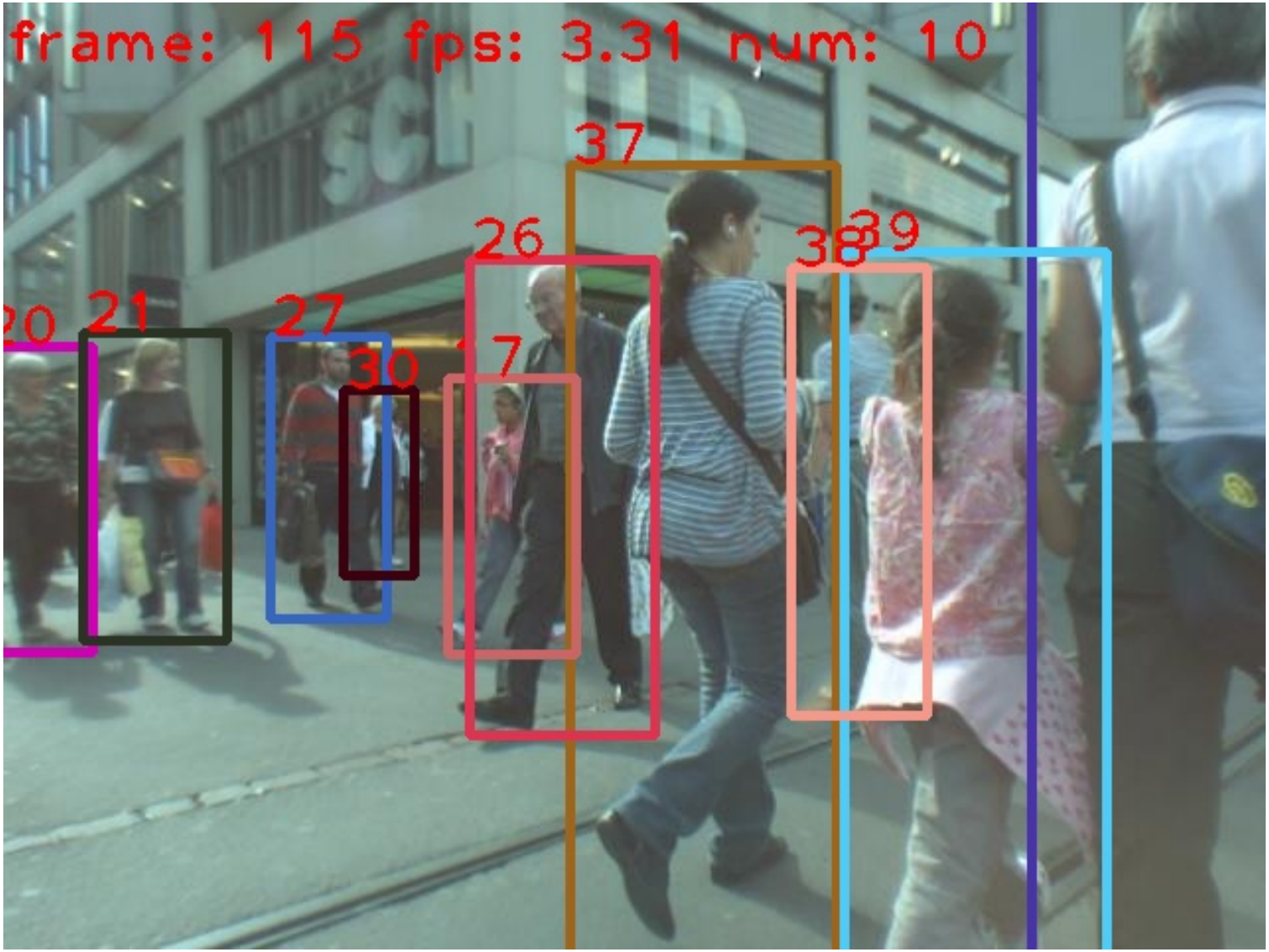
MOT Example: Base Heuristic



MOT Example: Base Heuristic

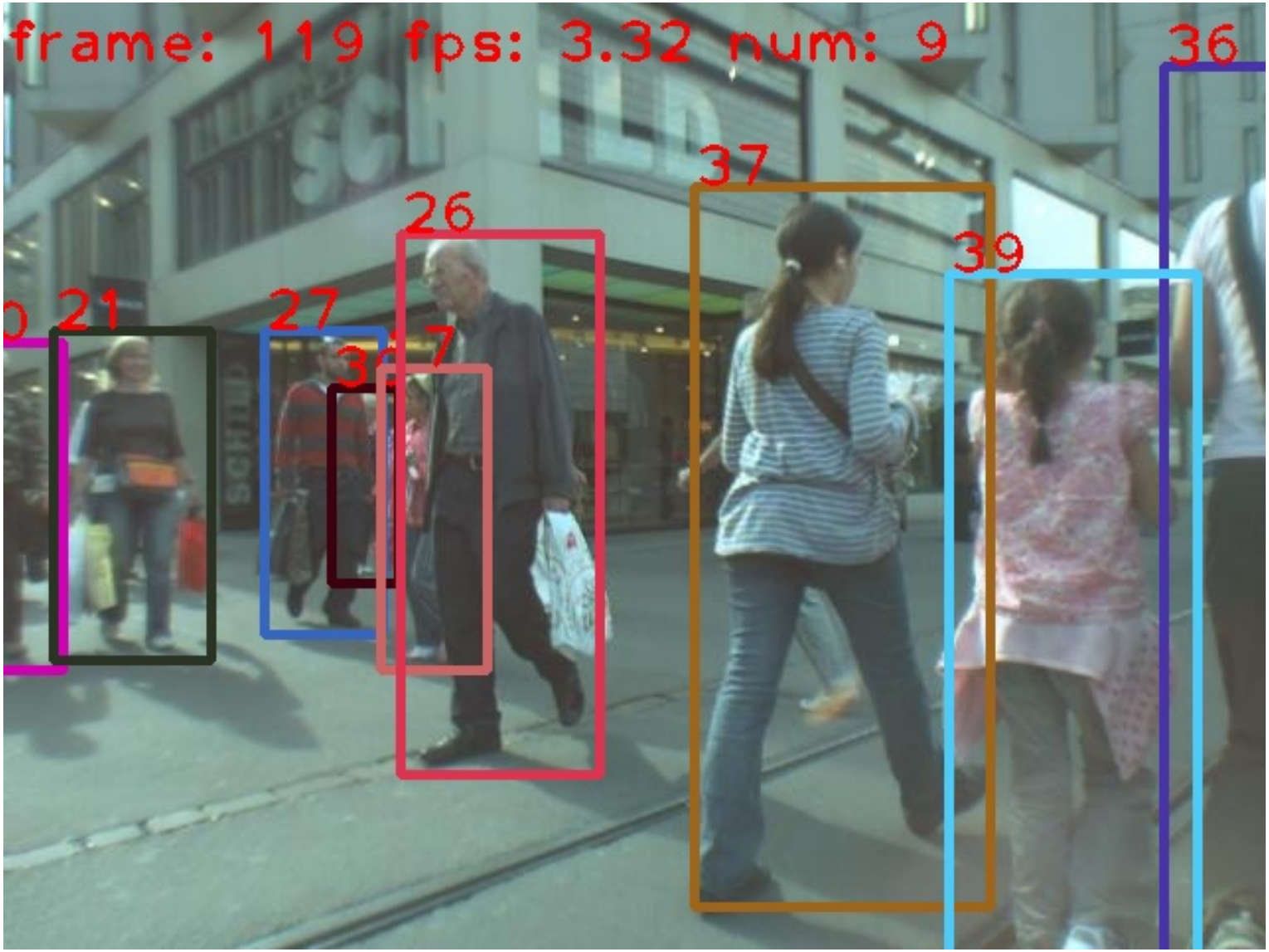


MOT Example: Base Heuristic

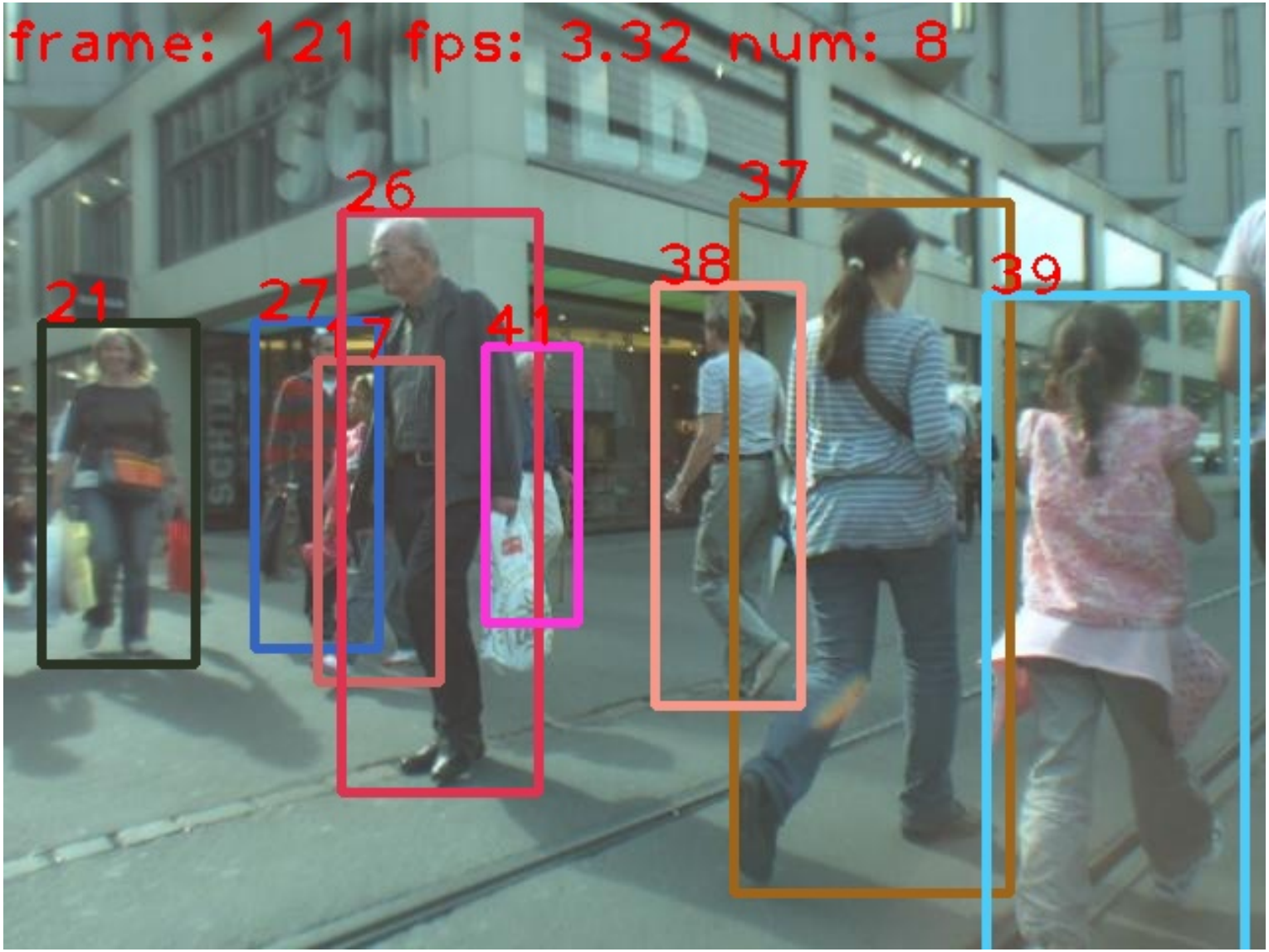


MOT Example: Base Heuristic

MOT Example: Base Heuristic



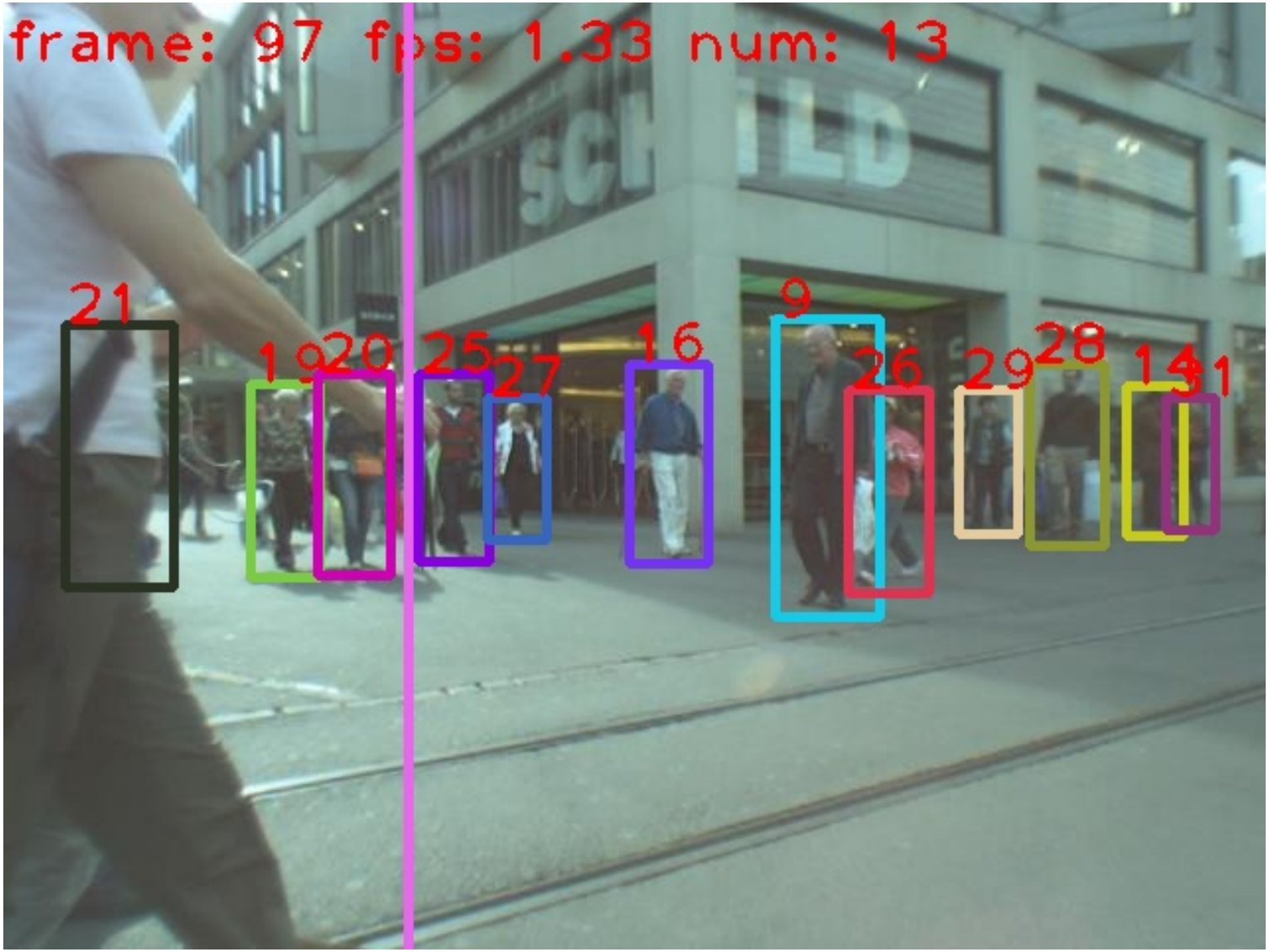
MOT Example: Base Heuristic



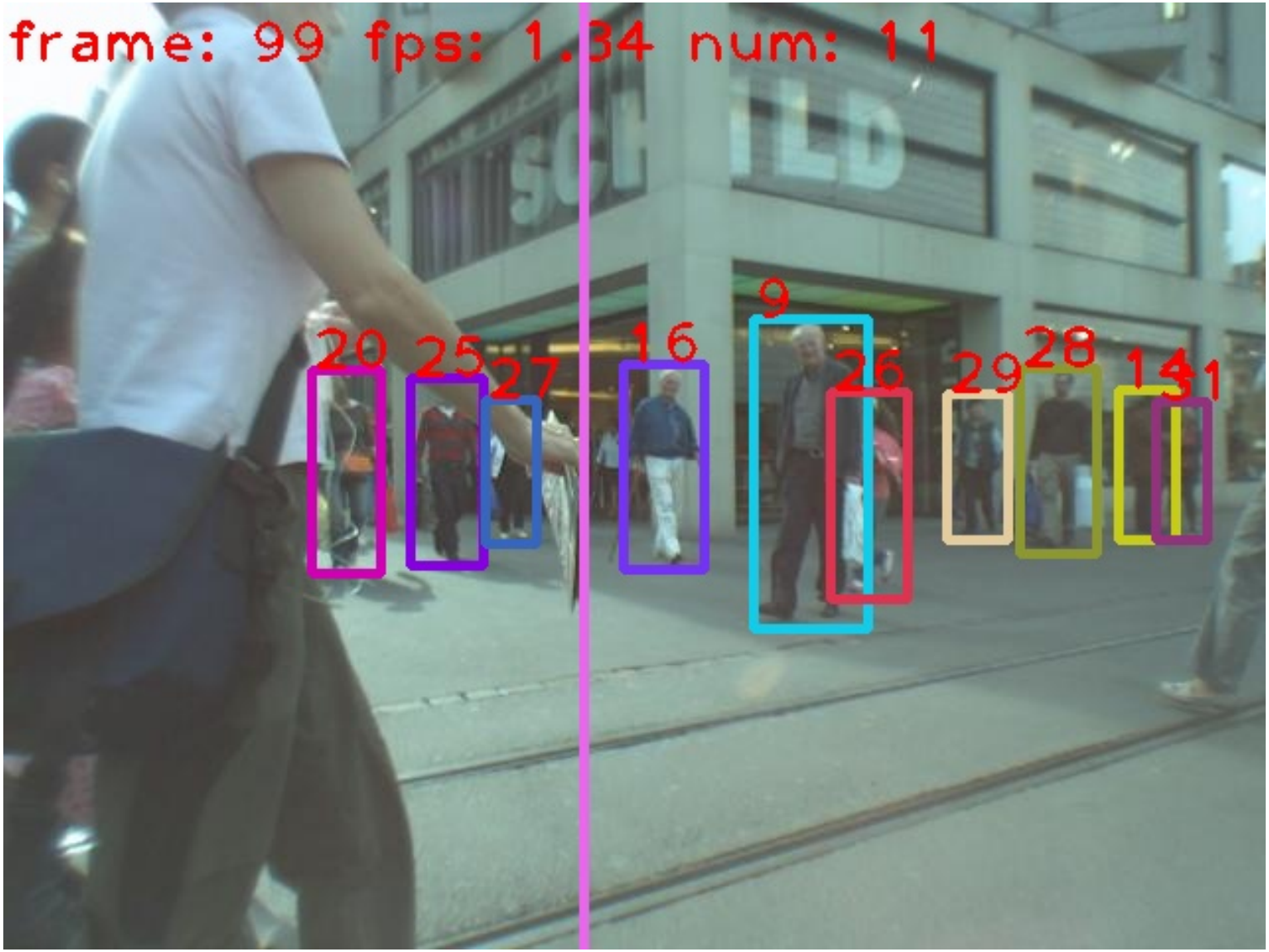
MOT Example: Base Heuristic

MOT Example: Base Heuristic

MOT Example: Approximation in Value Space



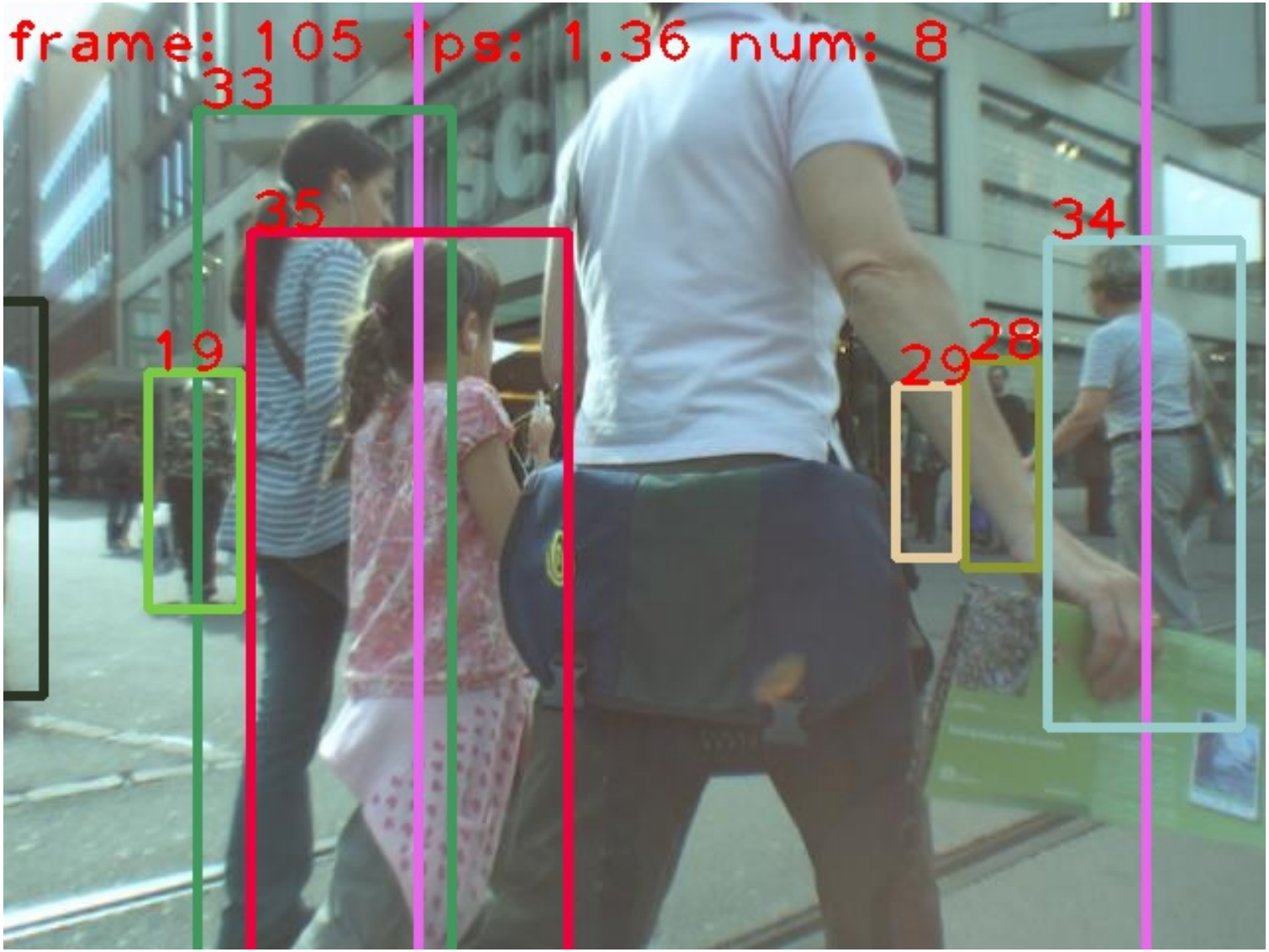
MOT Example: Approximation in Value Space



MOT Example: Approximation in Value Space

MOT Example: Approximation in Value Space

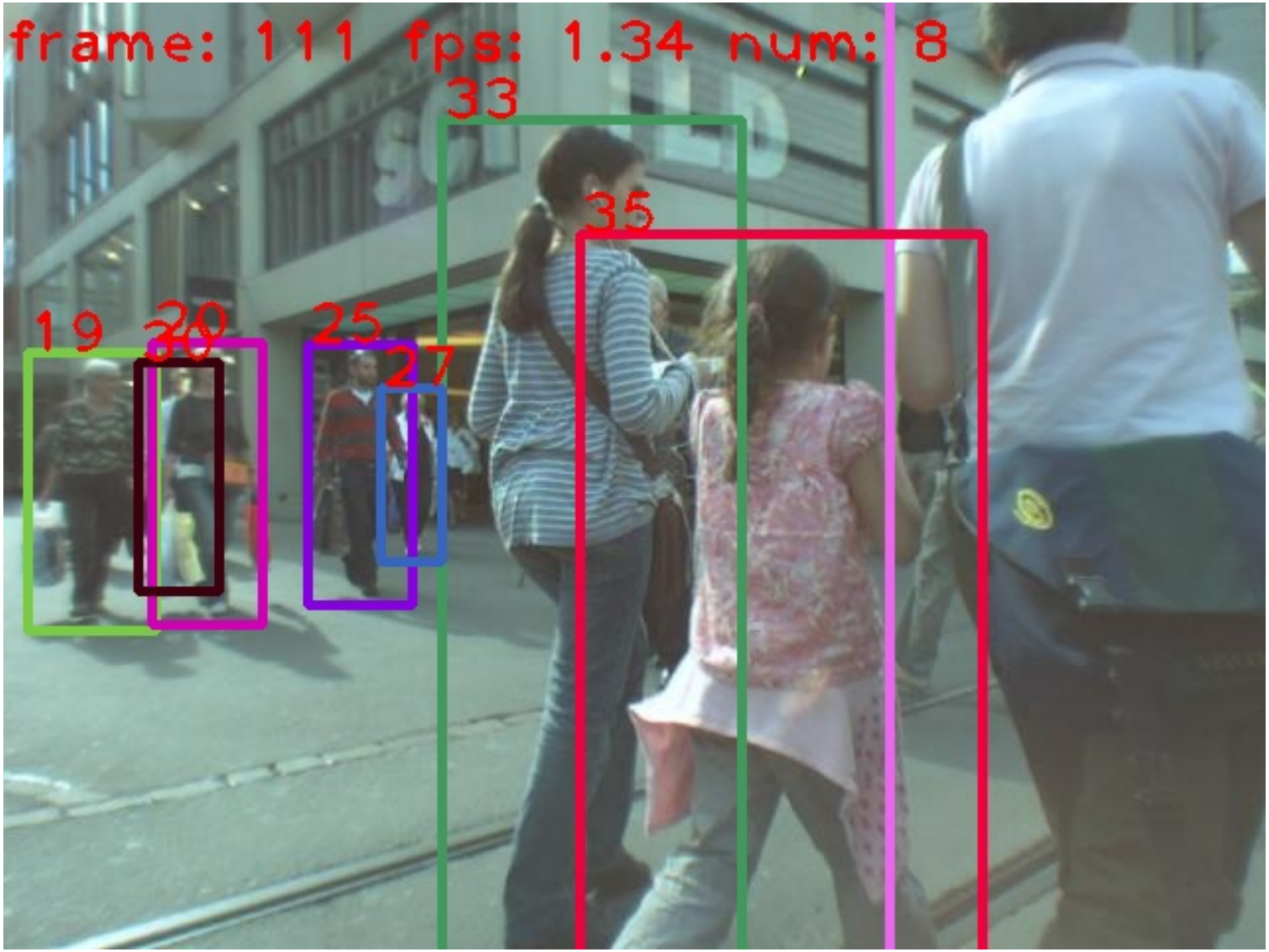
MOT Example: Approximation in Value Space



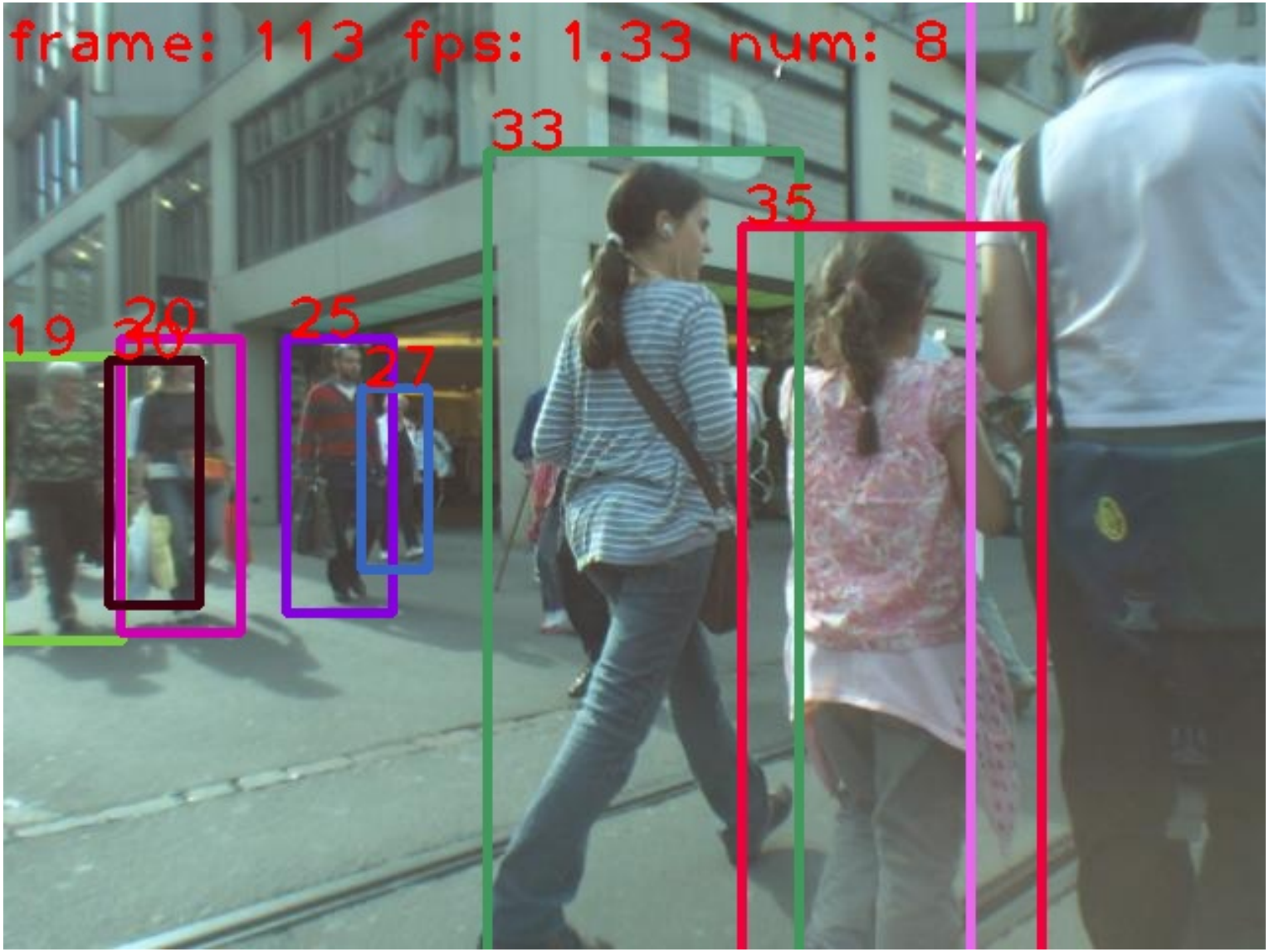
MOT Example: Approximation in Value Space

MOT Example: Approximation in Value Space

MOT Example: Approximation in Value Space



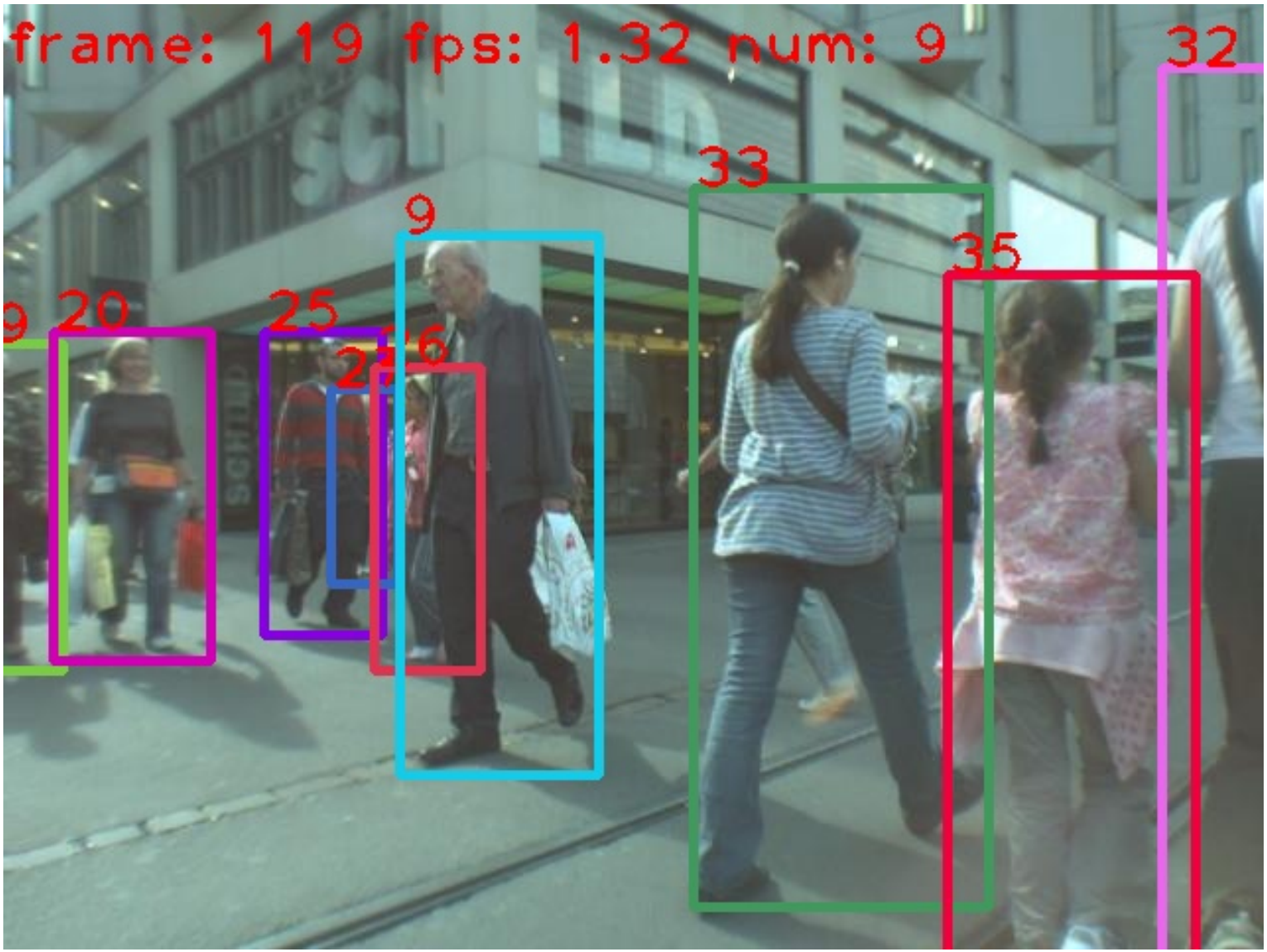
MOT Example: Approximation in Value Space



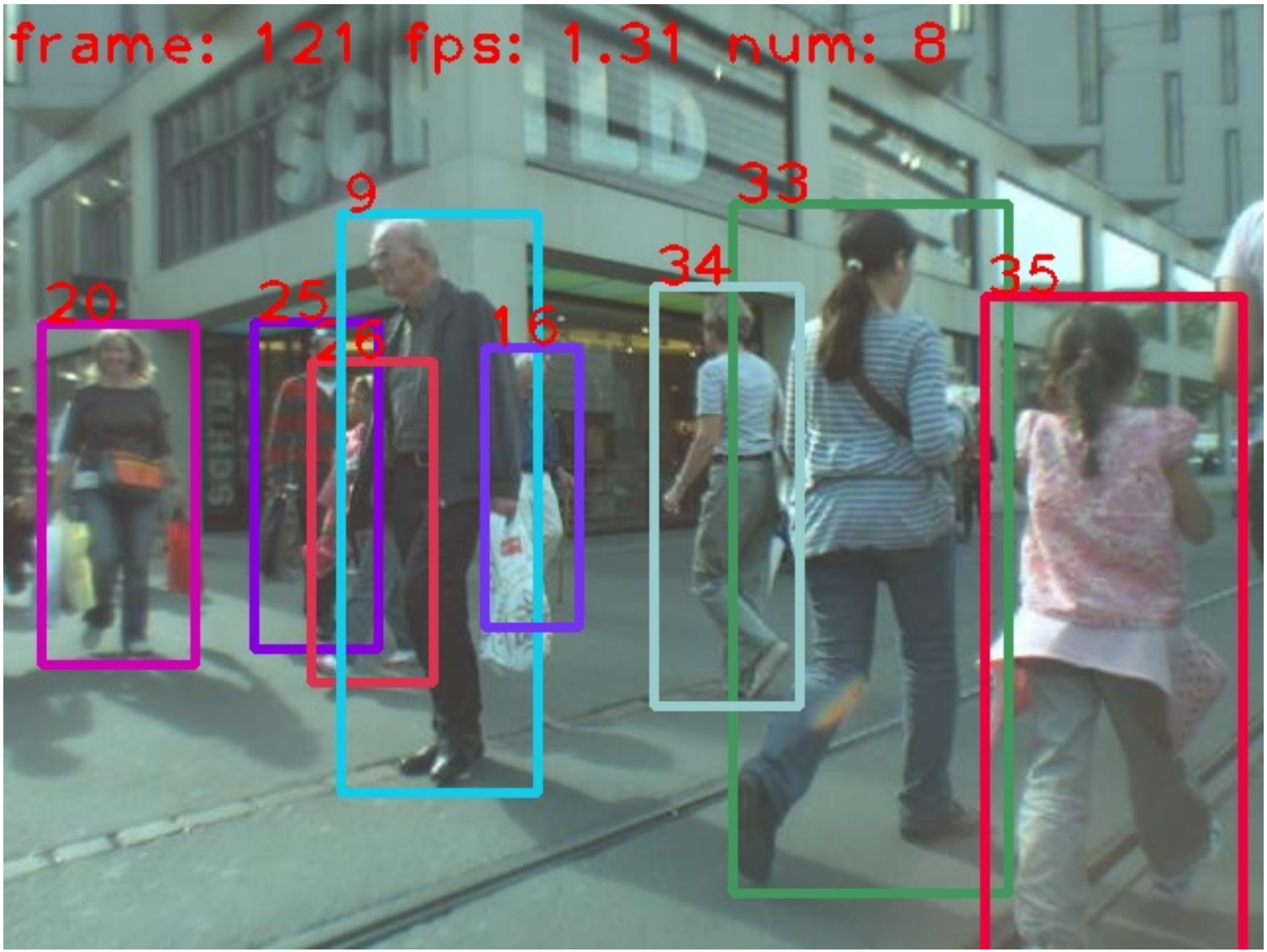
MOT Example: Approximation in Value Space

MOT Example: Approximation in Value Space

MOT Example: Approximation in Value Space



MOT Example: Approximation in Value Space



MOT Example: Approximation in Value Space

MOT Example: Approximation in Value Space