Environment-First Planning for Aggregate Transport: Foundations for Multi-Agent Systems

Nir Manor
Technion Autonomous Systems Program (TASP), Technion–Israel Institute of Technology
Haifa, Israel

Federico Oliva
Civil and Environmental Engineering, Technion – Israel Institute of Technology
Haifa, Israel

Amir Degani*
Civil and Environmental Engineering, Technion Autonomous Systems Program (TASP), Technion – Israel Institute of Technology
Haifa, Israel

ABSTRACT

Autonomous earthwork aims to reduce human labor in challenging construction environments by coordinating the movement of unlabeled aggregates through robotic manipulation and multi-agent path planning. This work proposes an *environment-centric* push-manipulation strategy for shovel-equipped rovers, replacing traditional pick-and-place methods designed for discrete items. The approach alternates between two modes: *Rearrange* (shaping aggregates into corridors) and *Deliver* (transporting material along corridors toward a target location).

To support scalable coordination, a *trajectory texture fabric* is introduced, consisting of a global set of trajectories computed on a discretized occupancy grid. From this fabric, a pushing heatmap $\mathcal{H}$ highlights high-flow "highway" paths that guide efficient aggregate transport. After each action, only the locally affected grid region is updated, and the heatmap and highways are recalculated, ensuring scalability. An optional spillage model accounts for aggregate loss during pushing, allowing for route re-evaluation to prioritize higher delivery efficiency.

Simulation results show that full trajectory computation scales efficiently with grid size, while selective regional updates significantly improve performance and maintain stable memory usage. A PyBullet interface confirms that 2D trajectory plans transfer reliably to a 3D rover model. By combining the trajectory texture fabric, pushing heatmap highways, and selective recomputation, we establish a reusable framework that agents can query in real time, enabling scalable and efficient autonomous earthwork systems.

KEYWORDS

Construction robotics, multi-agent systems, autonomous earthwork

ACM Reference Format:
Nir Manor, Federico Oliva, and Amir Degani. 2026. Environment-First Planning for Aggregate Transport: Foundations for Multi-Agent Systems. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 8 pages.

*Corresponding author: adegani@technion.ac.il

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus.

1 INTRODUCTION

Autonomous earthwork is an emerging field in construction robotics that seeks to reduce human exposure to harsh environments and increase productivity [1]. While traditional pick-and-place approaches are well-suited for handling single objects [2, 3], they become inefficient when moving large numbers of unlabeled aggregates such as soil, gravel, or debris.

Our work addresses high-throughput aggregate transport by embracing a *push-manipulation* paradigm. Instead of individually grasping objects, small rovers equipped with shovels push aggregates through the environment, shaping them toward a target region. This problem sits at the intersection of *robotic manipulation*, *earth-moving automation*, and *multi-agent path planning* [4–6]. Past research explored pushing and grasping in cluttered environments [5, 6], but few approaches tackle bulk material transport at scale.

Our key insight is to adopt an *environment-first perspective*. We compute a trajectory texture fabric over the terrain and derive heat maps that represent desirable movement directions for the material. This method decouples the environment plan from the exact trajectories of individual rovers, enabling both scalable single-agent operations and a pathway toward multi-agent coordination. By selectively updating only the affected regions after each push, we maintain high throughput without expensive global replanning.

Although we validate our framework using a single rover in 2D and 3D simulations, the environment-centric abstraction makes it amenable to coordination among multiple rovers. Recent advances in multi-agent path finding (MAPF) and distributed planning [7–10] can be integrated with our heat-map representation to handle collision avoidance, task allocation, and synchronization. Thus, this work lays the groundwork for multi-agent aggregate transport while delivering tangible benefits in the single-agent case.

1.1 Related Work

Non-prehensile manipulation and pushing. The mechanics of robot pushing were first formalized by Mason in 1986 [11]. Ben-Shahar and Rivlin [4] studied the rearrangement of movable objects, laying groundwork for later push–grasp frameworks. Dogar and Srinivasa [5] introduced a push–grasp planner for cluttered environments, while King et al. developed a rearrangement planner using both object-centric and robot-centric actions [3]. Cosgun et al. demonstrated efficiency gains by pushing multiple objects with a robot's forearm [2], and Zhou et al. optimized contact-rich manipulation under uncertainty [12]. Stüber et al. provided a comprehensive survey on robot pushing [6].

Aggregate transport and earth-moving. Handling bulk aggregates requires different strategies than those for single-object manipulation. Shitole et al. applied reinforcement learning to optimize earthmoving operations [13], and Nguyen et al. surveyed autonomous systems for volatile earth-moving [1]. Our earlier work proposed an aggregate-forming planner using convex hulls [14], and the AG2U method focused on grading under uncertainty [15]. Sun et al. introduced mean-shift exploration for swarm shape assembly [16], while Zhang et al. applied multi-task deep reinforcement learning to dynamic scheduling of earthmoving fleets [17].

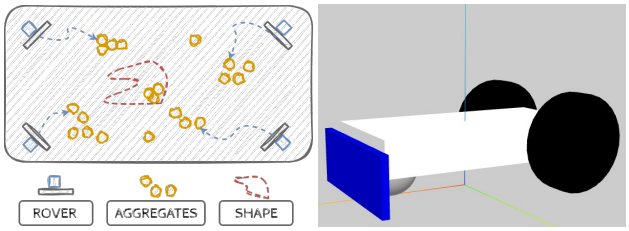
Multi-agent transport and path planning. Multi-agent path finding (MAPF) is a mature area with recent surveys [7] and a special track at JAIR [18]. Nordström et al. developed a risk-aware distributed MAPF algorithm [8], and Lindqvist et al. presented a distributed collision-avoidance scheme for UAVs [19]. Collision avoidance verification for multi-agent systems with neural feedback loops has been proposed by Dong et al. [20]. Peron et al. introduced a coordination and synchronization framework for multi-robot systems [9], and ARNL's project explores game-theoretic motion planning [21]. Game theory's role in multi-agent systems has also been discussed in the Milvus AI blog [22]. Hady et al. demonstrated multi-agent reinforcement learning for coordinated satellite tasks [10]. On the planning side, Li et al. proposed a self-learning Monte Carlo tree search (MCTS) algorithm [23], Xue et al. introduced tensor factorization to MCTS [24], and Riviere et al. presented spectral expansion MCTS for dynamical systems [25]. These methods provide scalable planning components that can be integrated with environment-centric aggregate transport.

1.2 Contributions

Inspired by [3], this study proposes an *environment-centric* approach, planning pushing actions from the perspective of the aggregates to be moved. Using a discretized representation of the environment, where aggregate positions are modeled via an occupancy grid, a global set of trajectories (i.e., a trajectory texture fabric) is constructed, optimizing push actions toward the target zone and summarized in a *pushing heatmap* $\mathcal{H}$. Analysis of the denser regions of this heatmap identifies optimal paths for moving aggregates to the target area. From this environment-centric description of potential pushing actions, each agent can query one of the following operational modes:

- *Rearrange*: Use $\mathcal{H}$ to plan heat-biased connectors that funnel aggregates from scattered cells onto highway paths H_i.
- *Deliver*: Traverse highway paths to the target zone $\mathcal{T}$.

The computationally intensive global path enumeration and field construction are performed once per environment update, depending solely on the environment state, not the number of agents. At execution, each agent performs a lightweight query, resulting in a near-constant marginal cost for adding agents. Additionally, this study addresses spillage, namely the loss of aggregates beyond the shovel boundaries during pushing. Incorporating a simple spillage model into material dynamics prediction enables a realistic evaluation of delivered mass, biasing planning toward trajectories with higher delivery yield.



(a) (b)

Figure 1: Task overview. Fig. 1a, illustrates agents pushing aggregates over a grid toward a target region; Fig. 1b depicts a single agent equipped with a shovel.

1.3 Notation and paper structure

Vector quantities use bold lowercase letters; (μ, σ) denote mean and standard deviation; $\|\cdot\|$ is the 2-norm; $\dot{x}$ is the time derivative of x. **Lexicographic scores:** whenever we write an expression such as

$$\arg\max_{P\in\mathcal{P}} \big(\, \mathrm{deliv}(P),\ -\mathrm{len}(P)\,\big),$$

the tuple indicates *priority order*: first maximize $\mathrm{deliv}(P)$; ties (equal delivery) are then broken by minimizing $\mathrm{len}(P)$ (because of the minus sign). More generally, $\arg\max(\alpha_1, \alpha_2, \dots)$ means lexicographic optimization on $(\alpha_1, \alpha_2, \dots)$.

This paper is organized as follows. Sec. 2 states the problem and the basic quantities. Sec. 3 details the method (visibility, search with continuation reuse, heat map, highways, and selective updates). Sec. 4 reports simulation results (2D/3D integration and benchmarks). Secs. 5 and 6 discuss future directions and conclude.

2 PROBLEM STATEMENT

This study addresses the autonomous collection and delivery of *unlabeled aggregates* (e.g., gravel, soil) into a user-specified target region $\mathcal{T}$ using a team of n mobile agents equipped with pushing tools, as introduced in Sec. 1. A schematic of the scenario and the agents is shown in Fig. 1. Each agent A_i is assumed to be localized, and the environment is monitored by a top-view camera capable of detecting aggregates.

Symbol	Meaning (used mainly in Sec. 3)
$\mathcal{G}$	Grid (set of all cells)
$\mathcal{T}$	Target region (deposition zone)
$\partial\mathcal{T}$	Boundary of the target region
$\mathcal{S}$	Occupied (non–target) cells with $\mathrm{obj}(C) > 0$
$\mathrm{obj}(C)$	Estimated aggregate count in cell C
$d(\cdot,\cdot)$	Euclidean distance (used to orient motion)
θ_{tol}	Global angular tolerance for forward motion
$P^{\star}_{\mathrm{target}}(C)$	Selected path from C to $\mathcal{T}$ (Sec. 3.2)
$u(C),\ \hat{v}(C)$	First successor on $P^{\star}_{\mathrm{target}}(C)$; unit direction
$\psi(C)$	Propagated raw mass used for aggregation (Sec. 3.3)
$H(\cdot), \mathcal{H}_{\tau_H}$	Heat map and derived highway set (Sec. 3.4)

Table 1: Main quantities and symbols used in the paper.

Remark 1 (Unlabeled aggregates). *Although identifiers are assigned to aggregates during the detection, the planning procedure described in Sec. 3 does not rely on persistent IDs. At each planner iteration, aggregates are re-detected without requiring consistent identifiers, allowing the method to handle generic unlabeled aggregates.*

State and discretization. The workspace is discretized into a square, axis-aligned grid $\mathcal{G}$, where each cell size approximately matches an agent's push footprint (as in [14]). Each cell $C \in \mathcal{G}$ stores an estimated aggregate count $\text{obj}(C) \geq 0$ obtained from perception (i.e., the top-view camera). Tracking individual grains is infeasible; instead, aggregates are represented at the cell level. All non-target cells (i.e., those not in $\mathcal{T}$) with $\text{obj}(C) > 0$ are considered *occupied*, forming the set $\mathcal{S}$.

Target region. The delivery area is defined as a compact planar set $\mathcal{T} \subset \mathbb{R}^2$ (e.g., a circle or polygon). During route computation (see Sec. 3), each occupied cell C generates a motion vector directed toward the boundary of $\mathcal{T}$, denoted $\partial\mathcal{T}$.

Sensing, observability, and uncertainty. At planning time, *full observability* is assumed: aggregate counts $\text{obj}(C)$ and agent poses are available from the top-view camera, represented as a 2D occupancy grid. Uncertainty arises instead from physical interactions, such as inter-aggregate collisions and shovel–aggregate dynamics. In particular, pushing may cause *spillage* (loss or dispersion) along a route. As introduced in Sec. 3, incorporating a spillage model biases planning toward higher-yield routes.

Planning objective. The primary objective is to maximize the volume of material delivered to $\mathcal{T}$. Secondary objectives aim to minimize transport cost (e.g., path length or makespan) and expected spillage. When multiple routes yield comparable delivery, these secondary criteria are used as tie-breakers.

3 METHOD

This section outlines the construction of an environment–centric pipeline for aggregate transport. The process begins by defining a forward–direction rule, along with the notion of *occupied* successor cells, which serve as the foundation for planning. For each occupied cell, a graph search is performed, guided by visibility constraints, to generate a short list of promising candidate routes toward the target region. These candidate routes are then aggregated into a look–back heat map, from which coherent *highway* corridors can be extracted. Finally, after each pushing action, only the portion of the map that has been affected is updated, ensuring computational efficiency while maintaining an accurate representation of the task.

3.1 Visibility Cone and Occupied Successors

From any cell C with geometric center c, candidate moves are biased toward the target region. Let $\hat{t}(C)$ denote the unit vector from c to the closest point on the target boundary $\partial\mathcal{T}$. A neighboring cell C' with center c' is considered *forward-feasible* if the step direction $\widehat{c' - c}$ aligns with $\hat{t}(C)$ within a tolerance angle θ_{tol}:

$$\hat{t}(C) \cdot \widehat{c' - c} \;\geq\; \cos\theta_{\text{tol}}. \tag{1}$$

An optional monotonicity condition can be enforced, requiring that the move not increase the distance to the target:

$$d(C', \mathcal{T}) \;\leq\; d(C, \mathcal{T}), \tag{2}$$

Algorithm 1 Simplified Occupied Successors $\text{Vis}(C)$

Input: Cell C, target boundary $\partial\mathcal{T}$, tolerance θ_{tol}, monotone flag
Output: List of forward-feasible occupied neighbors

```
Neighbors ← 8-connected occupied neighbors of C ▷ Initialize candidate set
t̂ ← unit vector from C center to closest point on ∂T
Feasible ← ∅ ▷ Initialize empty list for feasible neighbors
for each neighbor N ∈ Neighbors do ▷ Single-pass filtering
    d̂ ← unit vector from C center to N center
    if t̂ · d̂ ≥ cos θ_tol and (not monotonicity or d(N, T) ≤ d(C, T)) then
        Feasible ← Feasible ∪ {N}
    end if
end for
if Feasible = ∅ then ▷ Boundary fallback integrated
    return BoundaryFallback(C → ∂T)
end if
return Feasible ▷ Return feasible neighbors
```

where $d(C, \mathcal{T})$ denotes the Euclidean distance from c to $\mathcal{T}$. Planning expands only through *occupied* neighbors. The set $\text{Vis}(C)$ collects those neighbors that satisfy the forward-feasibility rule and, if enabled, the monotonicity constraint. If no such neighbor exists, a simple *boundary fallback* is introduced, directing motion toward the closest point on $\partial\mathcal{T}$. At this stage, line-of-sight checks for occlusion are omitted to keep the search procedure lightweight. The procedure presented in this paragraph is summarized in Alg. 1.

3.2 Full Paths to the Target

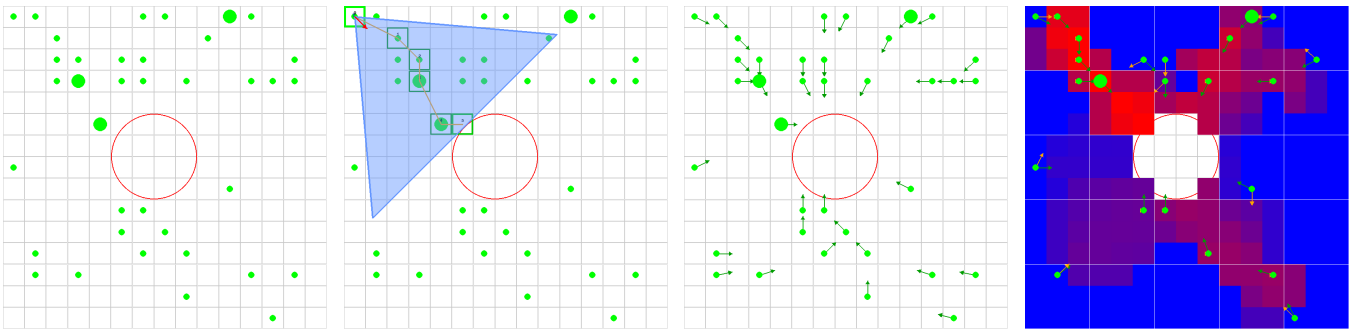
For each occupied source cell $C \in \mathcal{S}$, we run an A* search restricted to the occupied, forward-feasible neighbors (Sec. 3.1) in order to construct a *small set* of promising candidate routes from C to the target region. Rather than keeping only the single best path, we retain all routes whose delivered object count is within a slack margin τ of the current best. This ensures that near-optimal alternatives are preserved, which can later be useful when global considerations such as congestion or detours are taken into account. When two routes collect the same number of objects, the tie is broken in favor of the route with the shorter estimated total length. To prevent unrealistic paths, a global length cap can optionally be imposed. A pseudocode version of this procedure (including continuation reuse and near-optimal candidate retention) appears in Alg. 2.

Scoring (A with an optimistic look-ahead).* Each partial route ending at node v is scored using the A* evaluation function

$$f(v) \;=\; g(v) \;+\; h(v), \tag{3}$$

where the two terms are defined as follows:

- $g(v)$ represents the *raw object count* already accumulated along the path prefix leading to v.
- $h(v)$ is an *optimistic look-ahead estimate*, obtained by summing all objects currently visible from v within its forward cone, considering only cone-feasible occupied successors. This estimate deliberately ignores constraints such as cell capacity, depletion, double-counting, and spillage.



(a) Grid discretization. (b) Search to target (candidates). (c) First-successor directions $\hat{v}(\cdot)$. (d) Look-back heat map $H(\cdot)$.

Figure 2: Environment construction (four steps): (a) grid overlay of the raw scene; (b) visibility-aware search builds candidate routes to the target; (c) per-cell first-successor headings define a local velocity field; (d) look-back heat map from cached paths. *In (d),* green arrows at occupied cells show the heading from the selected target path $P^{\star}_{\text{target}}(C)$; orange arrows show via-highway connector headings. Background colors encode $H(X)$ (hotter = stronger incoming flow).

Since no actual route can collect more than this optimistic bound, $h(v)$ is an admissible heuristic: it safely overestimates the maximum remaining gain. This ensures that the A* search remains both efficient and correct, in the sense that it does not discard potentially optimal solutions. Ties in $f(v)$ are resolved by preferring shorter estimated path lengths

Continuation reuse (value substitution). Once the best continuation from a cell s to the target has been computed, the result is cached, and the heuristic at s is replaced by the exact pre-spillage raw yield of that continuation, denoted by $J^{\star}_{\text{raw}}(s)$. From that point on, any partial route ending at s is scored as

$$f(s) \;=\; g(s) \;+\; J^{\star}_{\text{raw}}(s). \tag{4}$$

When the search later encounters s, the current prefix is combined with the cached continuation to form a complete candidate route, avoiding any further expansion of s. Under the raw objective (with spillage disabled), selecting such an s with the largest f certifies global optimality and allows early search termination. When spillage is enabled, the cached continuation still provides a valid candidate, but it is re-ranked according to delivered mass rather than raw count. To maximize the reuse of cached continuations, sources are processed from near to far, so that continuations computed for closer cells are available to those farther away.

Per-source outputs and caching order. For each source cell C, the search produces a small candidate set of paths, denoted by $\mathcal{P}(C)$. From this set, one path $P^{\star}_{\text{target}}(C)$ is selected according to whether spillage is considered.

- **Spillage disabled.** Each candidate $P \in \mathcal{P}(C)$ is scored by the pair $(\text{raw\_objs}(P), -\text{len}(P))$, where $\text{raw\_objs}(P)$ is the number of collected objects and $\text{len}(P)$ is path length, i.e.,

$$P^{\star}_{\text{target}}(C) \;=\; \arg\max_{P\in\mathcal{P}(C)} \big(\,\text{raw\_objs}(P),\; -\text{len}(P)\,\big). \tag{5}$$

- **Spillage enabled.** For each $P \in \mathcal{P}(C)$, the delivered mass is computed by $\text{deliv}(P) = \text{Spill}(P;\Theta)$. Candidates are then scored by $(\text{deliv}(P), -\text{len}(P))$, yielding

$$P^{\star}_{\text{target}}(C) \;=\; \arg\max_{P\in\mathcal{P}(C)} \big(\,\text{deliv}(P),\; -\text{len}(P)\,\big). \tag{6}$$

After selection, two additional quantities are cached:
(1) the first successor $u(C)$ along $P^{\star}_{\text{target}}(C)$,
(2) the unit vector $\hat{v}(C) = \widehat{c \longrightarrow c_1}$, with c_1 the center of $u(C)$.

Spillage model. Spillage is evaluated by a lightweight, path conditioned function $\text{Spill}(P;\Theta)$ that can be swapped without changing the planner. In our prototype, the path P is first smoothed (spline) and local curvature κ_i is computed along the curve; spillage at sample i is proportional to both curvature and current load,

$$\text{spill}_i \;=\; \lambda\,\kappa_i \cdot \text{load}_i, \qquad \lambda \in \Theta,$$

with thresholds to ignore negligible losses. Delivered mass is the residual load at the target after subtracting accumulated spills, and total mass is conserved (delivered + spilled = picked up, up to rounding). This curvature-driven evaluator reflects the intuition that sharper turns spill more and straight segments spill less; any alternative evaluator can be used by plugging it into $\text{Spill}(\cdot;\Theta)$.

Propagated mass $\psi(\cdot)$ (upstream potential). Once one route has been selected for each occupied source, every route is assigned a single value: the number of objects that would reach the target *before* spillage. This value is then assigned onto every cell along the route. If a cell belongs to multiple routes, it retains the maximum of the values encountered. The resulting map $\psi(C)$ represents the upstream potential at cell C, i.e., how much flow could reach the target through C given the currently chosen routes. Formally, for a source R with selected path $P^{\star}_{\text{target}}(R)$, define

$$\Phi(R) \;=\; \text{raw\_objs}\big(P^{\star}_{\text{target}}(R)\big). \tag{7}$$

The propagated mass is initialized to zero, $\psi \equiv 0$, and updated as

$$\psi(U) \;\leftarrow\; \max\{\psi(U),\, \Phi(R)\}, \tag{8}$$

for every cell U on the path. This propagated mass $\psi(\cdot)$ is later used as the weighting factor in the heat-map computation.

3.3 Heat Map from Full Paths

The heat map $H(X)$ quantifies the strongest incoming flow at each cell X, based on the cached optimal paths $P^{\star}_{\text{target}}(\cdot)$ from Sec. 3.2. It aggregates contributions from source cells using the path-induced

Algorithm 2 Simplified A* Target Search

Input: Source cell C, successor function $\textsc{Vis}(\cdot)$, heuristic $h(\cdot)$, slack τ, spillage flag
Output: Candidate path set $\mathcal{P}$

Initialize priority queue $\textsc{Open}$ with $(C, g = 0, f = h(C))$ ▷ Custom comparator for ties
$\mathcal{P} \leftarrow \emptyset$, $r^\star \leftarrow -\infty$ ▷ Track candidates and best raw score
$\textsc{Cache} \leftarrow \emptyset$ ▷ Store continuations
while $\textsc{Open}$ not empty **do**
  $(v, g_v, f_v) \leftarrow \textsc{Open.pop}()$ ▷ Pop node with largest f, ties by shorter length
  if v in $\textsc{Cache}$ **then** ▷ Reuse cached continuation
    $P \leftarrow \text{prefix}(v) \oplus \textsc{Cache}(v)$
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{P\}$, $r^\star \leftarrow \max(r^\star, g_v + J^\star_{\text{raw}}(v))$
    if not spillage **then**
      return $\arg\max_{P \in \mathcal{P}}(\text{raw\_objs}(P), -\text{len}(P))$
    end if
    continue
  end if
  if spillage **and** $\max(f \in \textsc{Open}) < r^\star - \tau$ **then** ▷ Slack-based early termination
    break
  end if
  for $u \in \textsc{Vis}(v)$ **do** ▷ Expand feasible successors
    $g_u \leftarrow g_v + \text{objects}(u)$, $f_u \leftarrow g_u + h(u)$
    $\textsc{Open.push}(u, g_u, f_u)$
  end for
  if v is target **then** ▷ Cache continuation at target
    $\textsc{Cache}(v) \leftarrow \text{prefix}(v)$, $J^\star_{\text{raw}}(v) \leftarrow g_v$
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{\text{prefix}(v)\}$, $r^\star \leftarrow \max(r^\star, g_v)$
  end if
end while
return spillage? $\{P \in \mathcal{P} : \text{raw\_objs}(P) \geq r^\star - \tau\}$: $\arg\max_{P \in \mathcal{P}}(\text{raw\_objs}(P), -\text{len}(P))$

direction $\hat{v}(\cdot)$ and propagated mass $\psi(\cdot)$, facilitating the identification of high-traffic corridors.

Near-Field Gate and Directional Aggregation. For a cell X with center x and a source $C \in \mathcal{S}$ with center c and first successor $u(C)$ (center c_1), a source contributes to $H(X)$ only if it satisfies:

- *Near-field condition*: $\|x - c\| \leq \|c_1 - c\|$, ensuring X is closer to C than its first hop.
- *Forward-facing condition*: $\hat{v}(C) \cdot (x - c) > 0$, preventing backflow along the path direction $\hat{v}(C) = \frac{c_1 - c}{\|c_1 - c\|}$.

The alignment between the source's path direction and the vector to X is defined as:

$$a(C, X) = \max\left\{0, \hat{v}(C) \cdot \frac{x - c}{\|x - c\|}\right\}. \tag{9}$$

The heat map value is computed as:

$$H(X) = \max_{\substack{C \in \mathcal{S}: \\ \|x-c\| \leq \|c_1-c\|, \\ \hat{v}(C)\cdot(x-c) > 0}} a(C, X) \cdot \psi(C), \tag{10}$$

Algorithm 3 Heat Map Calculation (Look-Back with near-field-first gating)

Input: $\mathcal{S}$ (sources); $\hat{v}(\cdot)$ (path velocity); $\psi(\cdot)$ (propagated raw mass); first successor $u(\cdot)$ with center c_1 when defined
Output: Heat map $H(\cdot)$ on $\mathcal{G} \setminus \mathcal{T}$

for each $X \in \mathcal{G} \setminus \mathcal{T}$ with center x **do**
  $H(X) \leftarrow 0$
  for each $C \in \mathcal{S}$ with center c **do**
    if $u(C)$ **undefined then**
      continue ▷ no cached path/velocity
    end if
    if $\|x - c\| > \|c_1 - c\|$ **then**
      continue ▷ near-field gate (within first hop)
    end if
    if $\hat{v}(C) \cdot (x - c) \leq 0$ **then**
      continue ▷ forward-only (back-facing prune)
    end if
    $a \leftarrow \hat{v}(C) \cdot \frac{x-c}{\|x-c\|_2}$ ▷ alignment in $(0, 1]$
    $H(X) \leftarrow \max\{H(X),\ a \cdot \psi(C)\}$
  end for
end for
return $H(\cdot)$

where $\psi(C)$ is the propagated mass from Sec. 3.2. If no source satisfies the conditions, $H(X) = 0$. The maximum operator prioritizes the source with the strongest aligned flow, emphasizing corridors with high transport potential (see Alg. 3).

3.4 Highways and Highway Paths

We extract highway cells as a super-level set of the heat map,

$$\mathcal{H}_{\tau_H} = \left\{ X \in \mathcal{G} \setminus \mathcal{T} \,\middle|\, H(X) \geq \tau_H \right\}, \tag{11}$$

where τ_H is a user threshold (optionally after smoothing / non-max suppression to stabilize anchors).

Anchor choice (heat–distance trade-off). For a root (source) $R \in \mathcal{S}$ whose local heat is low, we pick a highway *anchor* $h \in \mathcal{H}_{\tau_H}$ by balancing highway strength against how far it is from R. In practice

$$h^\star = \arg\max_{h \in \mathcal{H}_{\tau_H}} \Big(H(h) - \lambda\, d(R, h) \Big), \tag{12}$$

with a user weight $\lambda \geq 0$ (equivalently one may use $H(h)/(1 + \mu\, d(R, h))$). This favors nearby anchors with strong flow while avoiding long detours to only slightly hotter cells.

Connector to the chosen anchor (simple scoring). Given $h^\star$, we plan a *connector* from R to $h^\star$ with the same graph search used for target paths, but oriented to the anchor: the forward direction at a cell C points to $h^\star$ ($\hat{h}(C) = \widehat{c \to h^\star}$), and the optional monotone rule checks $d(C', h^\star) \leq d(C, h^\star)$. Among feasible connectors P_{conn}, we prefer routes that *collect more while staying short*: first maximize the collected objects (pre-spillage raw count, or delivered mass if the spillage model is on), and use total length as a tie-break. The selected connector is then concatenated with the cached target suffix $P^\star_{\text{target}}(h^\star)$ to yield a via-highway route (see Alg. 4).

Algorithm 4 Finding and Using Highway Connectors

Input: Starting cell R, set of "highway" cells with high flow, map of heat values $H(\cdot)$, user parameter for distance trade-off λ
Output: Best connecting path from R to a highway

// **1. Pick a highway anchor**
for each highway cell h **do**
    Compute score = heat of h minus distance from R to h (balance these using λ)
end for
Pick h^* with the highest score
// **2. Find a path from** R **to** h^*
Plan a path that visits nearby cells toward h^*
Try to collect as many objects as possible on the way, but keep path short
// **3. Return the best path found**
The connector is the chosen path from R to h^*

3.5 Selective Rebuild After Execution

Executing a route modifies the object field along its cells and, if the spillage model is enabled, nearby spill-affected cells. To avoid recomputing the entire scene, updates are restricted to the affected region, followed by a single global refresh of aggregates.

Affected Cells. Let A_{exec} denote the set of cells with modified object counts due to route execution, including spill-affected cells when applicable. Sources that could be influenced by these changes are identified as:

$$A_{\text{vis}} = \left\{ C \in \mathcal{S} \mid \exists X \in A_{\text{exec}} \text{ with } \hat{t}(C) \cdot \frac{x-c}{\|x-c\|} \geq \cos\theta_{\text{tol}} \right\}, \quad (13)$$

where $\hat{t}(C)$ is the unit vector from the center c of source C to the closest point on the target boundary $\partial\mathcal{T}$, and θ_{tol} is the tolerance angle from Sec. 3.1. The affected region is defined as $A_{\text{all}} = A_{\text{exec}} \cup A_{\text{vis}}$. If $A_{\text{exec}} = \emptyset$, no updates are performed.

Selective Rebuild. A single iteration over A_{all} updates local quantities: visibility lists (Sec. 3.1), per-source paths, propagated mass $\psi(\cdot)$, and path directions $\hat{v}(\cdot)$ (Sec. 3.2). The execution path for each affected source is reselected. Then, global aggregates are updated:

(1) The heat map $H(\cdot)$ (Sec. 3.3) is recomputed once over the entire grid.
(2) The highway set $\mathcal{H}_{\tau_H}$ (Sec. 3.4) is updated based on the new heat map.
(3) Connector paths (Sec. 3.4) are replanned for sources whose anchors $h^\star$ change membership in $\mathcal{H}_{\tau_H}$ or whose connector paths intersect A_{all}.

If $A_{\text{vis}} = \emptyset$, only A_{exec} is updated locally, but the heat map is still recomputed to reflect changes in the object field.

Resulting Invariants. The rebuild ensures that:

(1) Cached paths and directions $\hat{v}(\cdot)$ are consistent with the updated object field.
(2) The heat map $H(\cdot)$ and highway set $\mathcal{H}_{\tau_H}$ reflect the updated caches.
(3) Recomputations are limited to A_{all}, except for the single global heat map update and minimal connector replanning.

Algorithm 5 Selective Rebuild After Execution

Input: Executed path(s), optional spillage map, tolerance θ_{tol}
Output: Updated caches: $P^\star_{\text{target}}(\cdot)$, $\psi(\cdot)$, $\hat{v}(\cdot)$, $H(\cdot)$, $\mathcal{H}_{\tau_H}$, $\mathcal{S}$

Identify affected cells:
    $A_{\text{exec}} \leftarrow$ cells on executed routes and spill-affected cells
    $A_{\text{vis}} \leftarrow \left\{ C \in \mathcal{S} \mid \exists X \in A_{\text{exec}} : \hat{t}(C) \cdot \frac{x-c}{\|x-c\|} \geq \cos\theta_{\text{tol}} \right\}$
    $A_{\text{all}} \leftarrow A_{\text{exec}} \cup A_{\text{vis}}$
if $A_{\text{exec}} = \emptyset$ **then return** ▷ No updates needed
end if
for each source $C \in A_{\text{all}}$, ordered by increasing $d(C, \mathcal{T})$ **do** ▷ Update local caches
    Recompute visibility list (Section 3.1)
    Recompute candidate paths using A* search with continuation reuse (Section 3.2)
    Select $P^\star_{\text{target}}(C)$, update $\hat{v}(C)$ from first hop, set $\psi(C)$
end for
Recompute heat map $H(\cdot)$ over $\mathcal{G}$ (Section 3.3)
Update highway set $\mathcal{H}_{\tau_H}$ from $H(\cdot)$ (Section 3.4)
Replan connectors for sources with anchors changed or paths intersecting A_{all} (Section 3.4)
Update $\mathcal{S}$: remove emptied cells, add newly occupied cells

This approach minimizes computational overhead while maintaining an accurate representation of the transport pipeline. Executing a route modifies the object field along its cells and, if the spillage model is enabled, nearby spill-affected cells. To avoid recomputing the entire scene, updates are restricted to the affected region, followed by a single global refresh of aggregates. A pseudocode summary appears in Alg. 5.

4 EXPERIMENTS

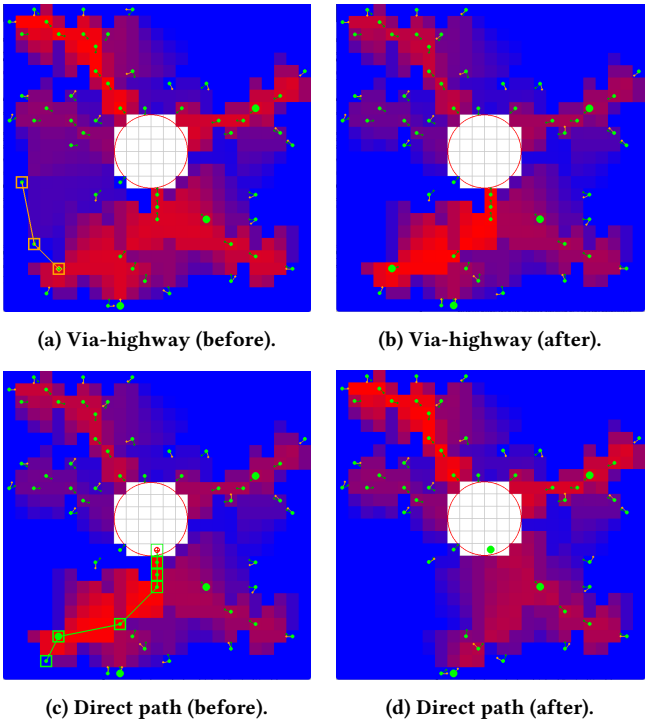
We evaluate the environment-centric pipeline along three axes: (i) compute scaling for *full* (from-scratch) calculations, (ii) per-tick efficiency of *Selective Updates*, and (iii) the impact of *continuation reuse* inside A*. We complement these with 2D/3D simulations that demonstrate execution fidelity.

4.1 Implementation and 2D Visualization

The planner is implemented in Python; 2D visualization uses `pygame`. Fig. 3 shows the two execution modes: the highway rearrangement and the direct path delivery.

4.2 3D Simulation Bridge

We also validate the 2D planner in PyBullet with a differential-drive rover and a front-mounted shovel. A light 3D↔2D bridge maps continuous world coordinates to the grid and back (Fig. 4): **(i)** shovel width sets the grid cell size; scene state (aggregate centroids, target, rover pose) is rasterized to the grid; **(ii)** the planner computes paths, the heat map, and (if needed) highway connectors; **(iii)** discrete waypoints are converted to smooth (spline-interpolated) 3D trajectories and executed; **(iv)** the updated 3D state (including spillage) is projected back to trigger Selective Updates.



(a) Via-highway (before). (b) Via-highway (after).

(c) Direct path (before). (d) Direct path (after).

Figure 3: Simulation snapshots. Heat map (hot=high) with per-cell headings (green) and connectors (orange). Top: connector to a highway corridor; Bottom: path to the target.

4.3 Benchmark Setup

Scenarios and protocol. Grids from 25×25 to 65×65 with varied object densities and seeds; runs are single-CPU, warmed, and measured on stabilized iterations (timers exclude I/O/plotting). Highways (anchors+connectors) are enabled but do not alter wall-time accounting for full vs. update passes.

Reporting. Unless noted, table entries are *medians across seeds*; values in parentheses are *95% bootstrap CIs.*

4.4 Benchmark Results

Table 2 summarizes performance. We also include a small linearity check (Fig. 5) showing *time per object* remaining essentially flat as object count increases.

What "19.5% improvement" means. When we disable continuation reuse inside A* and re-run the same instances, median runtime rises by 19.5%. Equivalently, *enabling* reuse reduces median runtime by 19.5% (Wilcoxon p=0.009), with parallel reductions in node expansions and peak open-set size.

4.5 Memory Footprint

Peak memory is ~410 MB (stable across scenarios). The *memory delta* per operation (After−Before) averages 1.48 MB and ranges from −7.4 MB (net free) to +14.7 MB (net allocate). Positive deltas correspond to transient allocations for search/updates/spillage; occasional negatives reflect GC/cleanup. Footprints are predictable with no leak-like growth.

4.6 Summary of Results

(1) Full recomputations scale smoothly with grid size and exhibit near-flat time per object;
(2) Selective Updates deliver large paired speedups (about 7×–9× faster across tested grids);
(3) Continuation reuse inside A* yields an additional 19.5% median runtime reduction;
(4) Memory use is modest and stable;
(5) The 3D bridge confirms that the 2D planning substrate executes cleanly in a physics simulator (single rover), supporting future multi-agent studies.

5 FUTURE WORK

Coordination atop the fabric. With the path fabric and highways in place, agents can be coordinated by MAPF-style planners [7, 8, 18] or market/game-theoretic mechanisms that reserve highway entries and time slots [21, 22]. Auction-based assignment over highway anchors is a natural fit for our "corridor" abstraction.

Advanced search integration. Continuation reuse and selective updates can warm-start *Monte Carlo Tree Search* (MCTS) planners [23–25], reducing rollout depth and improving sample efficiency. We will investigate hybrid planners that blend cached continuations with stochastic look-ahead.

Safety, timing, and verification. Distributed collision avoidance and synchronization frameworks [9, 19] can operate on our substrate by reacting to localized updates only; formal safety and verification with learned controllers [20] is another avenue.

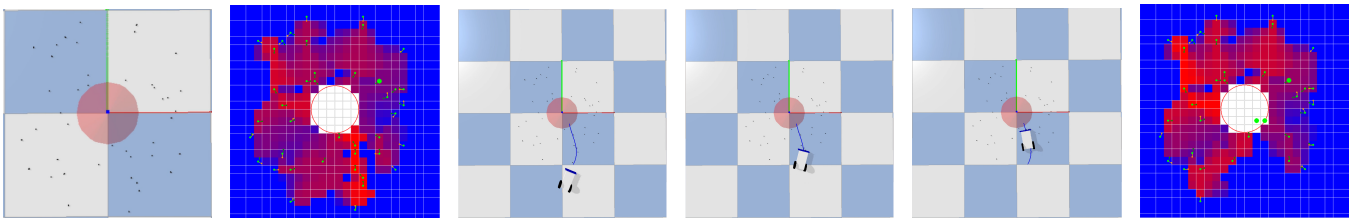
Learning on the substrate. The heat map and $\psi(\cdot)$ (raw flow potential) offer shaped rewards and stable features for multi-agent RL in scheduling/coordination [10, 17]. We will also explore learned spillage models as drop-in evaluators for candidate paths.

3D extension and partial observability. Extending the substrate to 3D terrains and relaxing full-observability (with online estimation) are engineering steps already underway.

6 CONCLUSION

We presented an environment–centric pipeline for transporting unlabeled aggregates to a target region. The method precomputes a trajectory fabric by running a visibility-aware A* from every occupied cell, using an admissible (optimistic) heuristic derived from cone-visible mass, and reusing exact continuations when a previously solved state is encountered. The resulting per-source paths yield (i) a path-induced direction field and (ii) a propagated raw-mass weight, which we aggregate via a look-back rule into a heat map whose ridges define highways. A selective update engine recomputes only affected regions after each execution, keeping wall-time low even as the number of agents grows.

Comparison to prior aggregate-forming work. Shaked et al.'s *Aggregate Forming Planner for Autonomous Earth-Moving* targets pushing based forming with a classical path-planning approach, emphasizing runtime, reduced computational intensity, and interpretability, and validates on simulation and an in-lab UR5e/UGV setup while handling spills between iterations [14]. In contrast, our contribution is not an execution-level controller but an *environmental substrate*: a precomputed, spillage-aware path fabric, a heat-map/highway



(a) 3D scene (b) 2D raster (c) Start (d) Mid (e) Approach (f) 2D after

Figure 4: 3D↔2D round-trip. 3D scene → 2D rasterization → 3D execution snapshots → updated 2D state after pushing.

Grid	**Full calc** (ms)			**Selective update** (ms)			**Speedup** S	**ms/object**
	Median	2.5%	97.5%	Median	2.5%	97.5%		Median
25×25	420	380	465	49	44	55	8.5×	17.9
45×45	980	910	1,060	129	118	143	7.6×	18.1
65×65	1,910	1,780	2,050	273	249	301	7.0×	18.4
Continuation reuse ablation (A):* enabling reuse reduces median runtime by **19.5%** (Wilcoxon p=0.009).								
Linearity (full calc): log–log slope $\alpha \approx 1.03$, $R^2 = 0.988$; time per object remains nearly constant within a grid.								

Table 2: Compact performance across grids and settings. Entries are *medians across seeds*; 95% bootstrap confidence intervals are given in the columns labeled 2.5% and 97.5%.

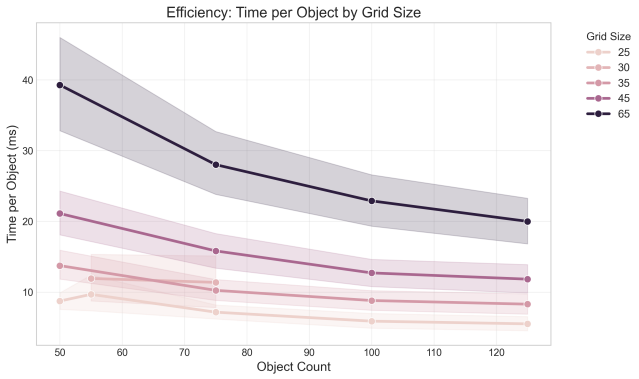


Figure 5: Linearity with object count. Time per object stays nearly constant within each grid size.

mechanism to funnel flow, and continuation reuse inside A* to reduce search work. This substrate is designed for many-agent use: agents query cached structures rather than triggering full replans, benefiting from selective updates and from the separation of *rearrangement* (into highways) and

REFERENCES

[1] Huy Nguyen, Ayanna Howard, and Melissa Johnson. Robotic autonomous systems for earthmoving equipment operating in volatile conditions and teaming capacity: A survey. *Robotica*, 40(9):2951–2975, 2022.

[2] Akansel Cosgun, Luke Ditria, Shayne D'Lima, and Tom Drummond. Embracing contact: Pushing multiple objects with robot's forearm. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[3] Jennifer E. King, Marco Cognetti, and Siddhartha S. Srinivasa. Rearrangement planning using object-centric and robot-centric action spaces. In *Robotics: Science and Systems*, 2016.

[4] Ohad Ben-Shahar and Ehud Rivlin. To push or not to push: On the rearrangement of movable objects by a mobile robot. Technical Report CIS-9802, Technion, 1998.

[5] Mehmet R. Dogar and Siddhartha S. Srinivasa. A framework for push-grasping in clutter. In *Robotics: Science and Systems (RSS)*, 2012.

[6] Jochen Stüber, Claudio Zito, and Rustam Stolkin. Let's push things forward: A survey on robot pushing. *Frontiers in Robotics and AI*, 7:8, 2020.

[7] Shiyu Wang, Haozheng Xu, Yuhan Zhang, et al. Where paths collide: A comprehensive survey of classic and learning-based multi-agent pathfinding, 2025.

[8] Samuel Nordström, Yifan Bai, Björn Lindqvist, and George Nikolakopoulos. A time-dependent risk-aware distributed multi-agent path planning, 2025.

[9] Davide Peron, Anna Pandita, Kevin Zhang, et al. Efficient coordination and synchronization of multi-robot systems under recurring linear temporal logic, 2025.

[10] Mostafa Hady, Ghina Al-Khaweja, Ayman Al-Ashwari, et al. Multi-agent reinforcement learning for autonomous multi-satellite earth observation: A realistic case study, 2025.

[11] M. T. Mason. Mechanics and planning of manipulator pushing operations. *The International Journal of Robotics Research*, 5(3):53–71, 1986.

[12] Jerry Zhou, Peter Florence, Lucas Manuelli, and Russ Tedrake. Convex optimization of contact-rich manipulation under uncertainty. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[13] Vivswan Shitole, Joseph Louis, and Prasad Tadepalli. Optimizing earth moving operations via reinforcement learning. 2019 Winter Simulation Conference, 2019. Conference paper.

[14] Tom Shaked, Karen Lee Bar-Sinai, Ari Meles-Braverman, Oren Elmakis, and Amir Degani. Aggregate-forming planner for autonomous earth-moving. *IEEE Access*, 11, 2023. Accepted 16 Oct. 2023; published online 23 Oct. 2023.

[15] Yakov Miron, Yuval Goldfracht, Chana Ross, Dotan Di Castro, and Itzik Klein. Ag2u: Autonomous grading under uncertainties, 2022. arXiv preprint.

[16] Jilin Sun, Yifan Wang, and Hui Chen. Mean-shift exploration in shape assembly of robot swarms. *Frontiers in Robotics and AI*, 10:1150938, 2023.

[17] Yunuo Zhang, Jun Zhang, Xiaoling Wang, and Tuocheng Zeng. Multi-task deep reinforcement learning for dynamic scheduling of large-scale fleets in earthmoving operations. *Automation in Construction*, 174:106123, 2025. Online first.

[18] Jair special track on multi-agent path finding. Journal of Artificial Intelligence Research (special track), 2024. Accessed 2025-10-08.

[19] Björn Lindqvist, Pantelis Sopasakis, and George Nikolakopoulos. A scalable distributed collision avoidance scheme for multi-agent uavs. Technical Report, 2021.

[20] Zihao Dong, Shayegan Omidshafiei, Michael Everett, et al. Collision avoidance verification of multiagent systems with neural feedback loops, 2024.

[21] Lasse Peters, Laura Ferranti, Javier Alonso-Mora, et al. Game-theoretic motion planning for multi-agent interaction. Project page, 2021.

[22] Milvus. What is the role of game theory in multi-agent systems? Blog post, 2025.

[23] Wei Li, Yi Liu, Ning Ding, and Yang Gao. A self-learning monte carlo tree search algorithm for robot path planning. *Frontiers in Neurorobotics*, 17:1039644, 2023.

[24] Teng Xue, Yan Zhang, Amirreza Razmjoo, and Sylvain Calinon. Monte carlo tree search with tensor factorization for robot optimization, 2025.

[25] Benjamin Riviere, John Lathrop, and Soon-Jo Chung. Monte carlo tree search with spectral expansion for planning with dynamical systems. *Science Robotics*, 2024. doi:10.1126/scirobotics.ado1010.