Distributed Transformations of Hamiltonian Shapes based on Line Moves

Abdullah Almethen, Othon Michail and Igor Potapov

Department of Computer Science
University Of Liverpool

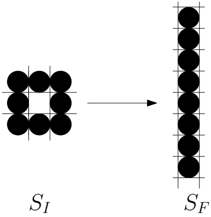
ALGOSENSORS 2021

Outline

Shape transformations

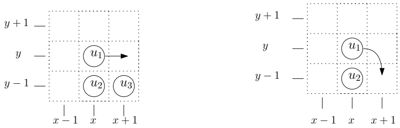
* Sometimes called *pattern formation*.
* One of the most essential goals for every robotic system.
* Given a 2D square grid.
* Each cell is occupied by a distinct device (agent) on the grid.
* Connected to each other and forming a shape S_I.
* Given a desired target shape S_F of the same order.

Goal: transform S_I into S_F via a finite number of valid moves.



Models of individual moves

- E.g., Dumitrescu *et al.* IJRR'04 and Michail *et al.* JCSS'19:
    - An individual can move over and turn around its neighbours.



- Akitaya *et al.* ESA'19 consider transformations based on similar moves.
- $\Omega(n^2)$ moves are required for all models of constant-distance individual moves.

Parallel transformations

- Multiple agents move together in a single time-step.
- Theoretical studies, such as Daymude *et al.*, Natural Computing'18.
- Practical implementations, such as Rubesntein *et al.*, Science'14.
- It can be shown that a connected shape can transform into any other connected shape, by performing in the worst case $O(n)$ parallel moves around the perimeter of the shape.

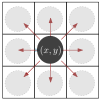
Models of more powerful mechanism

- Equip nodes with strong actuation mechanisms.
- Reduce the inherent distance by a factor greater than a constant in a single time-step.
- Linear-strength mechanisms, such as:
  - Aloupis *et al.*, (Computational Geometry'13) - an agent with arms can extend and contract its neighbours.
  - Woods *et al.*, (ITCS'13) - an agent has the ability to rotate a whole line of consecutive nodes.
  - Czyzowicz *et al.*, (ESA'19) consider a single moving robot that transforms a static shape by carrying its tiles one at a time.
  - Recently, we introduce the line-pushing model in (TCS'20).

The line-pushing model

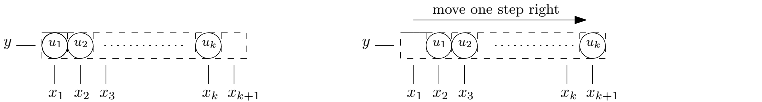
[Almethen et al., TCS'20]:

- An agent is connected to a neighbour vertically, horizontally or diagonally.



A node is *connected* to a neighbour at any directions.

- Equipped with a linear-strength mechanism which enables it to push a whole line in a single time-step, either vertically or horizontally but not diagonally.



The line-pushing model

- Exploiting the power of parallelism.
- Generalises the rotation and sliding models.
- Inherits all of their universality and reversibility properties.
- Allows diagonal connections on the grid.
- Achieved sub-quadratic time transformations,
  - An $O(n \log n)$-time universal transformation.
- All transformations are centralised,
  - Reveal the underlying transformation complexities.
- Though, not directly applicable to real robotic systems.

Contribution

- The first distributed transformation that exploits line moves within a total of $O(n \log n)$ moves, asymptotically equivalent to that of the best-known centralised transformations.
- Preserves all good properties of the centralised solutions.
- Include the *move complexity* (i.e. the total number of line moves).
- Also, its ability to preserve the connectivity.
  - Always guarantee that the graphs induced by the nodes occupied by the entities are connected during transformations.
  - An important assumption for many applications that usually require energy for:
    - Communication and data exchange.
    - Implementation of various locomotion mechanisms.

Technical challenges

– Several challenges must be overcome in order to develop such a distributed solution.

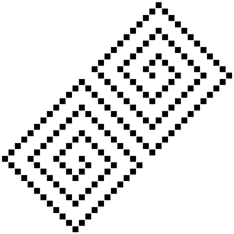
- Lack of global knowledge.
- Connectivity preservation.
- Timing - when to start/stop pushing
- Coordinating the moving of gnats, e.g:
  - Follow the same route.
  - No one is being pushed off.
- Agents do not automatically know whether they have been pushed.
  - It might be possible to infer this through communication and/or local observation.

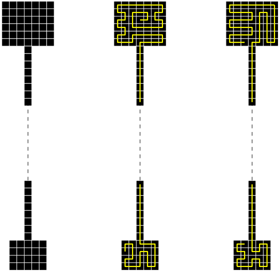
Distributed setting

- Discrete system of n indistinguishable agents on a 2D square grid.
- Agents act as finite-state automata, i.e.:
  - They have constant memory.
- Can observe the states of nearby agents in a Moore neighbourhood .
  - The eight cells surrounding an agent on the square gird.
- Operate in synchronised Look-Compute-Move (LCM) cycles.
- All communication is local.
- Actuation is based on:
  - Local information.
  - The agent's internal state.

An $O(n \log n)$-move Hamiltonian transformation

[Almethen *et al.*, ALGOSENSORS'20]:
– Introduced a connectivity-preserving strategy that transforms a pair of connected shapes (S_I, S_F) of the same order to each other.
– The associated graphs of both shapes contain a Hamiltonian path.





Problem Definition

– The proposed algorithm solves the line formation problem:

HamiltonianLine. Given any initial Hamiltonian shape S_I, the agents must form a final straight line S_L of the same order from S_I via line moves while preserving connectivity throughout the transformation.

– Preserves the best-known bound of $O(n \log n)$.

– A reasonable first step in the direction of more general distributed transformations in the given setting.

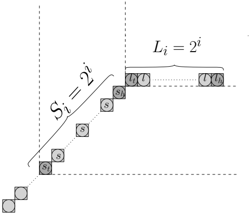
Definitions

- An agent $p \in S$ is defined as a 5-tuple (X, M, Q, δ, O), where
  - Q is a finite set of states.
  - X is the input alphabet for the states of the eight surrounding p on the grid, so $|X| = |Q|^8$.
  - $M = \{\uparrow, \downarrow, \rightarrow, \leftarrow, none\}$ is the output alphabet corresponding to the set of moves.
  - A transition function $\delta : Q \times X \rightarrow Q \times M$.
  - The output function $O : \delta \times X \rightarrow M$.
- A state $q \in Q$ of p is a vector with seven components $(c_1, c_2, c_3, c_4, c_5, c_6, c_7)$, where
  - c_1 is a label $\lambda \in \Lambda$ (p may be referred to by its label).
  - c_2 and c_3 are the transmission states.
  - c_4 and c_6 store a local direction $a \in A$.
  - c_5 holds a bit from $\{0, 1\}$.
  - c_7 is a pushing direction $d \in M$.

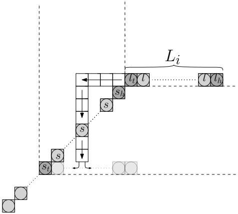
The Distributed Hamiltonian Transformation

- Assume a pre-processing phase provides the Hamiltonian path
- Proceeds in $\log n$ phases.
- Each phase consists of six sub-phases.
  - Every sub-phase running for one or more synchronous rounds.
- A Hamiltonian path P in S_I starts with a head labelled l_h,
  - Leads the process and coordinates all the sub-phases during the transformation.

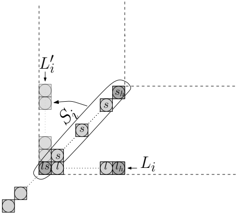
- Initially, the head l_h forms a trivial line of length 1.
- In each phase i, $0 \leq i \leq \log n - 1$, there exists a line L_i,
  - Starting from the head l_h,
  - Ending at a tail l_t,
  - With $2^i - 2$ internal agents labelled l in between.
- By the end of phase i, L_i will double its length via the six sub-phases.
  1. DefineSeg: Identify the next segment S_i of length 2^i.
  2. CheckSeg: Check the configuration of S_i.
  3. DrawMap: Compute a rout map that takes L_i to the end of S_i.
  4. Push: Move L_i along the drawn route map.
  5. RecursiveCall: A recursive-call to transform S_i into a straight line L_i'.
  6. Merge: Combine L_i and L_i' together into a straight line L_{i+1} of 2^{i+1} double length. Then, phase $i+1$ begins.



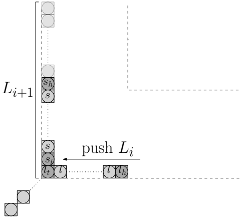
(a) DefineSeg, CheckSeg and DrawMap.



(b) Push.



(c) RecursiveCall.

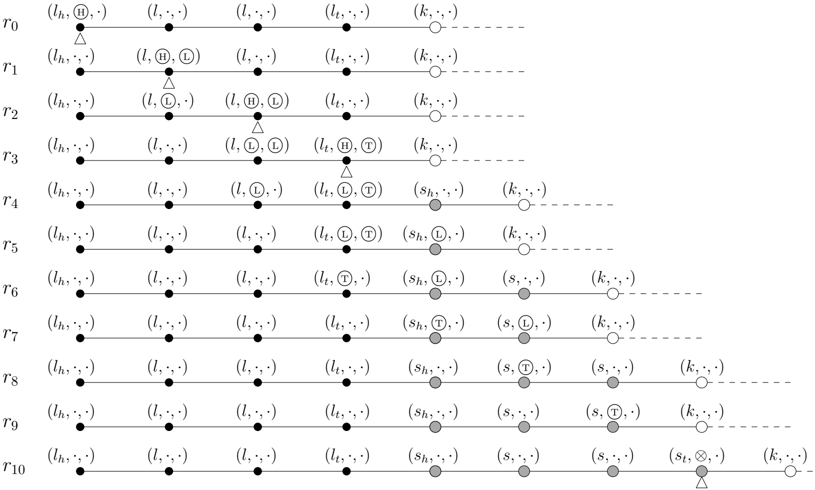


(d) Merge.

Defining the next segment S_i

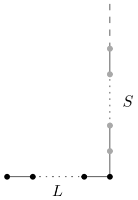
- Identifies the next 2^i agents on P.
- l_h emits a signal which is then forwarded by the agents along the line.
  - Moving from a predecessor p_i to a successor p_{i+1}.
  - Until it arrives at the first inactive agent, which becomes active.
- Similarly, each line agent initiates its own signal once it passes l_h's mark.
- Eventually, signals will re-label S_i, starting from a head in state s_h, has $2^i - 2$ internal agents in state s and ending at a tail s_t.

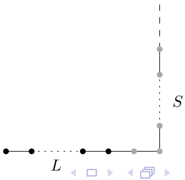
Defining the next segment S_i



Checking S_i

- Checks the S_i configuration, e.g. in line or perpendicular to L_i.
- A moving state initiated at L_i
- Checking each local direction relative to neighbours, which check each local direction relative to neighbours.
- If the check returns true, then
  - calls Merge to combine L_i and S_i into a new line L_{i+1} of length 2^{i+1}.
  - l_h starts a new round $i+1$.
- Otherwise, l_h proceeds with the next sub-phase, DrawMap.



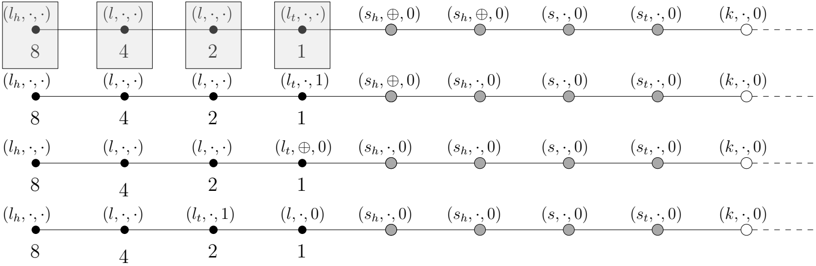


Drawing a route map

- l_h designates a route for L_i to push towards the tail s_t of S_i.
- Consists of two primitives:
  - ComputeDistance:
    - The line agents act as a distributed counter.
    - Compute Manhattan distance between the tails of L_i and S_i, $\Delta(l_t, s_t)$.
  - CollectArrows:
    - Local directions are gathered from S_i's agents.
    - Then distributed into L_i's agents.
    - Collectively draw the route map.
- Once this is done, L_i becomes ready to move and l_h can start the Push sub-phase.

Distributed Binary Counter

- l_h broadcasts a signal, asking all active agents to start the calculation.
- Once a segment agent p_i observes this signal, it emits
  - One increment mark "$\oplus$" if its local direction is cardinal.
  - Two sequential increment marks if it is diagonal.



CollectArrows procedure

– Draws a route that can be either

- Heading directly to s_t.
- Passing by the middle of S_i towards s_t.

– Segment agents then propagate their local directions stored in c_4 back towards l_h.

– Line agents distribute and rearrange S_i's local directions via several primitives, e.g.,

- Cancelling out pairs of opposite directions.
- Priority collection.
- Pipelined transmission.

– Finally, the remaining arrows cooperatively draw a route map.

CollectArrows procedure

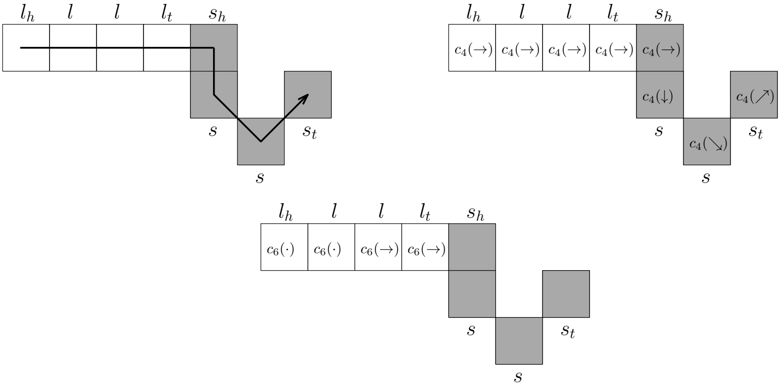


Figure: Drawing a map: from top-left a path across occupied cells and corresponding local arrows stored on state c_4 in top-tight, where the diagonal directions, "↘" and "↗", are interpreted locally as, "↓→" and "↑→". The bottom shows a route map drawn locally on state c_6 of each line agent.

Pushing L_i

- l_h synchronises with l_t to guide line agents during pushing.
- l_t moves simultaneously with l_h according to local direction $\hat{a} \in A$ in c_6.
- l_t checks the next cell (x, y) that L_i pushes towards whether
  - It is empty:
    - l_h pushes L_i one step towards (x, y).
    - All line agents shift their map arrows in c_6 towards l_t.
  - Occupied by an agent $p \notin L_i$:
    - Each line agent swaps states with p.
    - Tells l_h to push one step.
    - Until the line completely traverses the route and restores it to its original state.

Pushing L_i

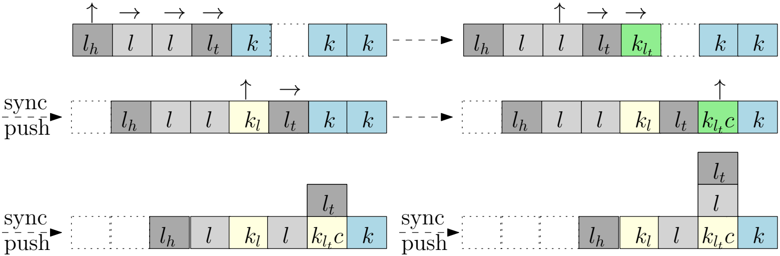
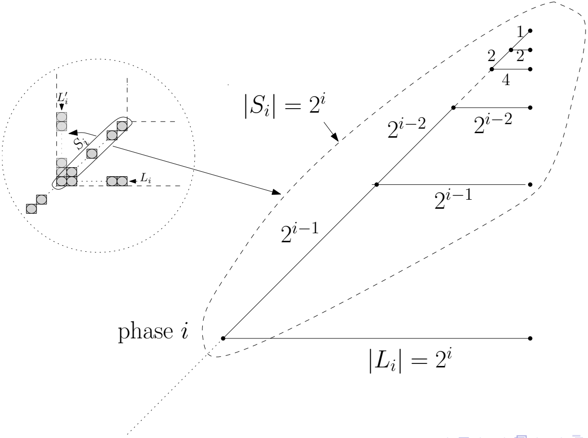


Figure: A line L_i of agents inside grey cells, with map directions above, pushing and turning through empty and non-empty cells in blue (of label k). The green and yellow cells show state swapping.

Recursive call on S_i

– l_h calls RecursiveCall to apply the general procedure recursively on S_i in order to transform it into a line L'_i.



Merging L_i and L'_i

– The final sub-phase of this transformation. – The agents of L_i and L'_i combine into a new straight line L_{i+1} of 2^{i+1}.
– Then, the head l_h of L_{i+1} begins a new phase $i+1$.

– Thus, we conclude that the call of RecursiveCall in the final phase $i = \log n$ requires a total moves:

$$T=\sum_{i=1}^{\log n} T(i)=\sum_{i=1}^{\log n} 2^{i-1}(i-1)-2^i=\sum_{i=1}^{\log n-1}(i-2)2^i-2^{\log n}\leq\sum_{i=1}^{\log n-1} i\cdot 2^i-n$$
$$\leq\sum_{j=1}^{\log n}\sum_{i=j}^{\log n} 2^i-n\leq\sum_{j=1}^{\log n} n-n\leq n\log n-n\leq O(n\log n).$$

Theorem

The above distributed transformation solves HAMILTONIANLINE *and takes at most $O(n\log n)$ line moves and $O(n^2\log n)$ rounds.*

Questions?