Introduction

- They investigated the problem of planning optimal paths for multiple robots with individual goals.

- The robots have identical but non-negligible sizes, are confined to some arbitrary connected graph, and are capable of moving from one vertex to an adjacent vertex in one time step.

- Collision between robots is not allowed, which may occur when two robots attempt to move to the same vertex or move along the same edge in different directions.
Introduction (Contd…)

- They propose a network flow based integer linear programming (ILP) model for finding robot paths that are time optimal or distance optimal

- The time optimality criterion seeks to minimize the number of time steps until the last robot reaches its goal

- The distance optimality seeks to minimize the total distance (each edge has unit distance) traveled by the robots
MULTI-ROBOT PATH PLANNING ON GRAPHS

Problem Formulation

• Let $G = (V,E)$ be a connected, undirected, simple graph (i.e., no multi-edges)

• $V = \{v_i\}$ is its vertex set

• $E = \{(v_i,v_j)\}$ is its edge set

• Let $R = \{r_1, \ldots, r_n\}$ be a set of robots that move with unit speeds along the edges of $G$

• Initial and goal locations on $G$ given by the injective maps $x_I, x_G : R \rightarrow V$, respectively

• The set $R$ is effectively an index set.

• A path or scheduled path is a map $p_i : Z^+ \rightarrow V$, in which $Z^+ := N \cup \{0\}$

• Intuitively, the domains of the paths are discrete time steps.
A path $p_i$ is feasible for a single robot $r_i$ if it satisfies the following properties

1. $p_i(0) = x_i(r_i)$

2. For each $i$, there exists a smallest $k_i^{min} \in \mathbb{Z}^+$ such that for all $k \geq k_i^{min}$, $p_i(k) = x_g(r_i)$

3. For any $0 \leq k < k_i^{min}$, $(p_i(k), p_i(k+1)) \in E$ or $p_i(k) = p_i(k+1)$

- Two paths $p_i, p_j$ are in collision if there exists $k \in \mathbb{Z}^+$ such that $p_i(k) = p_j(k)$ (collision on a vertex, or meet) or $(p_i(k), p_i(k+1)) = (p_j(k+1), p_j(k))$ (collision on an edge, or head-on).

- If $p(k) = p(k+1)$, then the robot stays at vertex $p(k)$ between the time steps $k$ and $k+1$. 
• **Problem 1:** *(MPP on Graphs)*

Given \((G, R, x_I, x_G)\), find a set of paths \(P = \{p_1, \ldots, p_n\}\) such that \(p_i\)'s are feasible paths for respective robots \(r_i\)'s and no two paths \(p_i, p_j\) are in collision.
Problem (Contd...)

- A natural criterion for measuring path set optimality is the number of time steps until the last robot reaches its goal. This is sometimes called the makespan, which can be computed from \( \{k_i^{\text{min}}\} \) for a feasible path set \( P \) as

\[
T_P = \max_{1 \leq i \leq n} k_i^{\text{min}}
\]

- Another frequently used objective is distance optimality, which counts the total number of edges traveled by the robots.
Illustration : Distance Optimality and Time Optimality

• Generally distance optimality and time optimality cannot be satisfied at the same time

• In above Fig let the dotted straight line have length \( t \) and the dotted arc has length \( 1.5t \) from some large even number \( t \)

• The four solid line segments are edges with unit length.

• Assuming that robot 1, 2 are to move from the locations marked with solid circles to the locations marked with gray dotted circles.

• Time optimal paths take \( 1.5t+2 \) time steps with a total distance of \( 2.5t +4 \); distance optimal paths take \( 2t +3 \) time steps with a total distance of \( 2t +4 \).

• In this paper, authors work with graphs on which the only possible collisions are meet or head-on collisions.
A Motivating Example

- We call this problem a 9-puzzle, which is a variant of the 15-puzzle.
MULTI-ROBOT PATH PLANNING AND MULTIFLOW

Network Flow

• A network $N = (G,c_1,c_2,S)$ consists of a directed graph $G = (V,E)$ with $c_1,c_2 : E \to \mathbb{Z}^+$ as the maps defining the capacities and costs on edges, respectively.

• $S = S^+ \cup S^-$, with $S^+$ denoting the set of sources and $S^-$ denoting the set of sink vertices.

• For a vertex $v \in V$, let $\delta^+(v)$ (resp. $\delta^-(v)$) denote the set of edges of $G$ going to (resp. leaving) $v$.

• A feasible (static) $S^+,S^-$-flow on this network $N$ is a map $f : E \to \mathbb{Z}^+$ that satisfies edge capacity constraints,

$$\forall e \in E, \ f(e) \leq c_1(e)$$
MULTI-ROBOT PATH PLANNING AND MULTIFLOW

Network Flow (Contd…)

• The flow conservation constraints at non terminal vertices

$$\forall v \in V \setminus S, \sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) = 0,$$

• The flow conservation constraints at terminal vertices

$$F(f) = \sum_{v \in S^+} \left( \sum_{e \in \delta^-(v)} f(e) - \sum_{e \in \delta^+(v)} f(e) \right) = \sum_{v \in S^-} \left( \sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) \right).$$

• The quantity $F(f)$ is called the value of the flow $f$.

• The classic (single-commodity) maximum flow problem asks the question: Given a network $N$, what is the maximum $F(f)$ that can be pushed through the network?
• The minimum cost maximum flow problem further requires the flow to have minimum total cost among all maximum flows

$$\sum_{e \in E} c_2(e) \cdot f(e).$$
The above formulation concerns a single commodity, which corresponds to all robots being inter exchangeable.

For MPP, the robots are not inter exchangeable and must be treated as different commodities.

Multi-commodity flow or multiflow captures the problem of flowing different types of commodities through a network.

Instead of having a single flow function $f$, we have a flow function $f_i$ for each commodity $i$.

The constraint become

$$\forall i, \forall e \in E, \sum_i f_i(e) \leq c_1(e),$$

$$\forall i, \forall v \in V \setminus S, \sum_{e \in \delta^+(v)} f_i(e) - \sum_{e \in \delta^-(v)} f_i(e) = 0,$$

$$\forall i, \sum_{v \in S^+} \left( \sum_{e \in \delta^-(v)} f_i(e) - \sum_{e \in \delta^+(v)} f_i(e) \right) = \sum_{v \in S^-} \left( \sum_{e \in \delta^+(v)} f_i(e) - \sum_{e \in \delta^-(v)} f_i(e) \right).$$
Equivalence between MPP and multiflow

- Viewing robots as commodities, one may connect MPP and multiflow
- To make the presentation clear, use as an example the simple graph $G$ with initial locations $\{s^+_i\}, i=1,2$ and goal locations $\{s^-_i\}, i=1,2$.

- An instance of Problem 1 is given by $(G, \{r_1, r_2\}, x_I: r_i \rightarrow s^+_i, x_G: r_i \rightarrow s^-_i)$
Equivalence between MPP and multiflow (Contd...)

- Convert the problem to a network flow problem, \( N' = (G', c_1, c_2, S^+ U S^-) \)
- Given the graph \( G \) and a natural number \( T \), create \( 2T + 1 \) copies of vertices from \( G \), with indices \( 0, 1, 1', \ldots \)
- For each vertex \( v \in G \), denote these copies \( v(0) = v(0), v(1), v(1'), v(2), \ldots, v(T) \).
- For each edge \((u, v) \in G \) and time steps \( t, t + 1, 0 \leq t < T \), add the gadget shown in Fig. between \( u(t), v(t) \) and \( u(t+1), v(t+1) \).
Equivalence between MPP and multiflow (Contd…)

• For the gadget, we assign unit capacity to all edges, unit cost to the horizontal middle edge, and zero cost to the other four edges. This gadget ensures that two robots cannot travel in opposite directions on an edge in the same time step.

• To finish the construction of Fig., for each vertex $v \in G$, we add one edge between every two successive copies (i.e., we add the edges $(v(0),v(1))$, $(v(1),v(1))$, \ldots , $(v(T),v(T))$).

• These correspond to the green and blue edges in Fig.

• For all green edges, assign them unit capacity and cost; for all blue edges, assign them unit capacity and zero cost.

• The network $N = (G,c_1,c_2,S^+U S^-)$ is now complete;
• Problem 1 has reduced to an integer maximum multiflow problem on $N$ with each robot from $R$ as a single type of commodity.
Equivalence between MPP and multiflow (Contd…)

- **Theorem 2**: Given an instance of Problem 1 with input parameters \((G,R,x_I,x_G)\), there is a bijection between its solutions (with maximum number of time steps up to \(T\)) and the integer maximum multiflow solutions of flow value \(n\) on the time-expanded network \(N\) constructed from \((G,R,x_I,x_G)\) with \(T\) time steps.
ALGORITHMIC SOLUTIONS FOR OPTIMAL MULTI-ROBOT PATH PLANNING

- Given the time-expanded network \( N = (G, c_1, c_2, S^+, S^-) \), it is straightforward to create an integer linear programming (ILP) model with different optimality objectives.

- They investigate two objectives in this section:
  1. Time optimality or makespan (the time when the last robot reaches its goal)
  2. Distance optimality (the total distance traveled by all robots).
**Time optimality**

- Time optimal solutions to Problem 1 can be obtained using a maximum multiflow formulation.

- As a first step, introduce a set of \( n \) loopback edges to \( G \) by connecting each pair of corresponding goal and start vertices in \( S \), from the goal to the start.

- For convenience, denote these loopback edges as \( \{e_1, \ldots, e_n\} \) (e.g., edges \( e_1, e_2 \)).

- These edges have unit capacity and zero cost.

- Next, for each edge \( e_j \in G' \), create \( n \) binary variables \( x_{i,j}, \ldots, x_{n,j} \) corresponding to the flow through that edge, one for each robot. \( x_{i,j} = 1 \) if and only if robot \( r_i \) passes through \( e_j \) in \( G' \).
Time optimality (Contd…)

• The variables $x_{i,j}$'s must satisfy two edge capacity constraints and one flow conservation constraint,

$$\forall e_j \in G', \quad \sum_{i=1}^{n} x_{i,j} \leq 1$$

$$\forall 1 \leq i, j \leq n, i \neq j, \quad x_{i,j} = 0,$$

$$\forall v \in G' \text{ and } 1 \leq i \leq n, \quad \sum_{e_j \in \delta^+(v)} x_{i,j} = \sum_{e_j \in \delta^-(v)} x_{i,j}.$$  

• The objective function is

$$\max \sum_{1 \leq i \leq n} x_{i,i}.$$
Time optimality (Contd…)

- For each fixed \( T \), the solution to the above ILP problem equaling \( n \) means that a feasible solution to Problem 1 is found.
- Find the minimal \( T \) that yields such a feasible solution.
- To do this, start with \( T \) being the maximum over all robots the shortest possible path length for each robot, ignoring all other robots.
- We then build the ILP model for this \( T \) and test for a feasible solution.
- If the model is not feasible, we increase \( T \) and try again
- The first feasible \( T \) is the optimal \( T \).
- The algorithm is complete: Since the problem is discrete, there is only a finite number of possible states. Therefore for some sufficiently large \( T \), there must either be a feasible solution or we can pronounce that none can exist.
- Calling this algorithm MINMAKESPAN (time optimal MPP)

**Proposition 3** Algorithm MINMAKESPAN is complete and returns a solution with minimum makespan to Problem 1 if one exists.
Distance optimality

- Distance optimality objective can be encoded using minimum cost maximum multiflow.
- To force a maximum flow, let $x_{p,i} = 1$ for $1 \leq i \leq n$.
- The objective is given by:

$$\min \sum_{e_j \in G', j > n, 1 \leq i \leq n} c_2(e_j) \cdot x_{i,j}.$$ 

- The value given when feasible, is the total distance of all robots’ paths.
- Let $T_i$ denote the optimal $T$ produced by MINMAKESPAN (if one exists), then a distance optimal solution exists in a time-expanded network with $T = nT_i$ steps.
- Calling this algorithm MINTOTALDIST (distance optimal MPP).
- Proposition 4: Algorithm MINTOTALDIST is complete and returns a solution with minimum total path length to Problem 1 if one exists.
Due to the large number of steps needed in the time expanded network, MINTOTALDIST, in its current form, is not very fast in solving problems with many robots. 

Therefore, our evaluation in this paper focuses on MINMAKESPAN which, on the other hand, is fairly fast in solving some very difficult problems. 

MINTOTALDIST, however, still proves useful in providing time optimal and near distance optimal solutions using the outputs of MINMAKESPAN.
PROPERTIES OF THE $n^2$-PUZZLE

• The example problem from Fig. 2 easily extends to an $n \times n$ grid; call this class of problems the $n^2$-puzzle.

• Such problems are highly coupled: No robot can move without at least three other robots moving at the same time.

• At each step, all robots that move must move synchronously in the same direction (per cycle) on one or more disjoint cycles

• Proposition 5 All states of the 9-puzzle are connected via legal moves.

• Larger puzzles can be solved recursively

• First solve the top and right side of the puzzle and then the left over smaller square puzzle.
PROPERTIES OF THE $n^2$-PUZZLE

For a 16-puzzle, outlines the procedure, consisting of six main steps:

1. Move robots 1 and 2 to their respective goal locations, one robot at a time (first 1, then 2).
2. Move robots 3 and 4 (first 3, then 4) to the lower left corner (top-middle figure).
3. Move robots 3 and 4 to their goal location together via counterclockwise rotation along the cycle indicated in the top-middle figure in Fig. 8.
4. Move robot 8 to its goal location.
5. Move robots 12 and then 16 to the lower left corner.
6. Rotate robots 12 and 16 to their goal locations.
PROPERTIES OF THE $n^2$-PUZZLE

• **Proposition 6:** All states of an $n^2$-puzzle, $n \geq 3$ are connected via legal moves.

• **Corollary 7:** All instances of the $n^2$-puzzle, $n \geq 3$, are solvable.
SOLUTIONS AND EVALUATION

- Experimentation in this paper focuses on MINMAKESPAN with the main goal being evaluating the comparative efficiency of the approach rather than pushing for best computational performance.

- As such, implementation is Java based and did not directly take advantage of multi-core technology.

- Gurobi, the ILP solver used for implementation, can engage multiple cores automatically for hard problems.

- Ran code on an Intel Q6600 quad-core machine with a 4GB JavaVM.
Time optimal solution to $n^2$-puzzles

- The first experiment performed was evaluating the efficiency of the algorithm MINMAKESPAN for finding time optimal solutions to the $n^2$-puzzle for $n = 3, 4, 5, \text{ and } 6$.

- Ran Algorithm MINMAKESPAN on 100 randomly generated $n^2$-puzzle instances for $n=3, 4, 5$.

- For the 9-puzzle, computation on all instances completed successfully with an average computation time of 1.36 seconds per instance.

- To compare the computational result, they implemented a (optimal) heavily optimized BFS algorithm.

- Since the state space of the 9-puzzle is small, the BFS algorithm is capable of optimally solving the same set of 9-puzzle instances with an average computation time of about 0.89 seconds per instance.

- Once we move to the 16-puzzle, the power of general ILP solvers becomes evident. MINMAKESPAN solved all 100 randomly generated 16-puzzle instances with an average computation time of 18.9 seconds.

- On the other hand, the BFS algorithm with a priority queue that worked for the 9-puzzle ran out of memory after a few minutes.
Time optimal solutions for grid graphs

- For problems in which not all graph vertices are occupied by robots, MINMAKESPAN can handle much larger instances.

- In a first set of tests on this subject, a grid size of $20 \times 15$ is used with varying percentage of obstacles (simulated by removed vertices) and robots for evaluating the effect of these factors.

- A typical set up is illustrated in Fig. 11. The computation time (in seconds) and the average number of optimal time steps (in parenthesis) are listed in Table I. The numbers are averages over 10 randomly created instances.

- For each run, a maximum of 1000 seconds is allowed (such limits, somewhat arbitrary, were chosen to manage the expected running time of the entire set of experiments; our complete algorithms should terminate eventually).

- Entries with superscript numbers suggest the 10 runs did not all finish within the given time.

- The superscript numbers represent the successful runs on which the statistics were computed. “N/A” means no instance finished within the allowed time.
Fig. 11. A 20 $\times$ 15 grid with 20% vertices removed (modeling obstacles) and 30 start/goal pairs. The start locations are marked with strings beginning with “S” and the goal locations are marked with strings beginning with “G”.
### TABLE I

<table>
<thead>
<tr>
<th>% obs</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2.5(22)</td>
<td>7.3(24)</td>
<td>16.7(27)</td>
<td>23.6(26)</td>
<td>70.7(27)</td>
</tr>
<tr>
<td>10</td>
<td>2.1(21)</td>
<td>7.8(24)</td>
<td>13.1(26)</td>
<td>20.4(26)</td>
<td>48.6(26)</td>
</tr>
<tr>
<td>15</td>
<td>3.9(25)</td>
<td>6.2(24)</td>
<td>13.8(26)</td>
<td>32.8(27)</td>
<td>126(28)</td>
</tr>
<tr>
<td>20</td>
<td>2.4(24)</td>
<td>7.7(27)</td>
<td>21.9(28)</td>
<td>39.3(26)</td>
<td>173(27)</td>
</tr>
<tr>
<td>25</td>
<td>2.7(27)</td>
<td>8.1(28)</td>
<td>24.8(30)</td>
<td>68.0(28)</td>
<td>253(30)</td>
</tr>
<tr>
<td>30</td>
<td>3.0(31)</td>
<td>29.9(34)</td>
<td>234(44)</td>
<td>80.6(29)</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### TABLE II

<table>
<thead>
<tr>
<th>% obs</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>14.4(41)</td>
<td>34.6(45)</td>
<td>43.7(44)</td>
<td>87.5(47)</td>
<td>402(49)</td>
</tr>
</tbody>
</table>
Conclusions

- Paper introduced a multiflow based ILP algorithm for planning optimal, collision-free paths for multiple robots on graphs.

- They provided complete ILP algorithms for solving time optimal and distance optimal MPP problems.

- Experiments confirmed that MINMAKESPAN is a feasible method for planning time optimal paths for tightly coupled problems as well as for larger problems with more free space.

- Moreover, MINMAKESPAN can serve as a good heuristic for solving large problem instances efficiently.