Discrete Optimization for Multi-Agent Path Finding

Peter J. Stuckey

Pierre Le Bodic, Graeme Gange, Daniel Harabor, Edward Lam, Jiaoyang Li, Sven Koenig
Outline

- Multi Agent Path Finding (MAPF)
  - Why should we be interested in this problem
  - MAPF Solving Methods
- CP for MAPF: Lazy CBS
  - CBS Weaknesses
  - Lazy CBS
  - Experiments
- MIP for MAPF: Branch and Cut and Price (BCP)
  - MIP Background
  - A MIP model of MAPF
  - Path Planning in BCP
  - Experiments
- Conclusion: What do Path Planning and Discrete Optimization have to say to each other
Multi-agent Pathfinding

- Consider an infinite time horizon with discrete time steps
- Consider a grid called the map
  - The map is divided into cells
  - Cells can be passable or obstacles
Multi-agent Pathfinding

- Consider a set of **agents**
- Every agent needs to move from its start cell to its end cell
Multi-agent Pathfinding

- At any time step, an agent can **move north, south, east** or **west**, or **wait** at the same cell.
Multi-agent Pathfinding

- **Vertex collision:** two or more agents cannot occupy a cell at any given time
Multi-agent Pathfinding

- **Swap (edge) collision**: two or more agents cannot cross in opposite directions at any given time
Multi-agent Pathfinding

- Minimise sum of individual costs (SIC) (sum of path lengths)
  - alternate makespan (maximum path length) is much easier

- Assumptions:
  - Centralized Solver
  - Offline Task
  - Optimal Solution
Why should we be interested in MAPF
The future of road travel

- From Peter Stone's website
  http://www.cs.utexas.edu/~pstone/
The present of road travel

- Uncontrolled intersection in Ethiopia
- https://youtu.be/SQ3rxwVYs5c
Motivation

- Robotics
- Video games
- Transportation applications
- Warehouse management
- Product assembly
Multi-agent Pathfinding

• The quintessential multi-agent movement coordination problem

  • Real world problems are harder but always have this underlying difficulty

• MAPF is NP-hard

• Who needs optimality?

  • Problems that arise for the optimal version

    • Also hit the bounded sub-optimal case

• Unbounded sub-optimal solutions can be very bad!
MAPF Solving Methods
MAPF Solving Approaches

- A* (and variations e.g. M*)
- Conflict Based Search (and variations)
- Encoding/Reduction
  - SAT, ASP, CP, MIP
- Suboptimal methods (we won't talk about)
  - ECBS
  - Push and Swap
  - Pebble rotation coordination (complete)
A* Approach

- **State space**: Permutations of $n$ agents into $V$ locations = $O(V^n)$
- **Operators**: Locations of all agent in the next time step
- **Heuristic function**: Sum of Individual Costs (SIC)

Even for this tiny example $m^2$ state space
Conflict Based Search

- Plan each agent individually
- repeat
  - if the plans for pair of agents $A, B$ conflict (use resource $X$ at same time $t$)
    - split the problem into two
      - left child: $A$ cannot use $X$ at time $t$
      - right child: $B$ cannot use $X$ at time $t$
    - pick a subproblem to work on
    - replan for the affected agent (enforcing the constraints)
  - else return plans
CBS Example

- Initial Plans
- Conflict at location C
- Two subproblems: mouse \{1,2\} cannot use C at time 2

\[
\langle m_1, C, 2 \rangle \quad \langle m_2, C, 2 \rangle
\]
CBS Example

- Select first subproblem
- Replan mouse 1: avoiding C at time 2
- Solution!

\[
\langle m_1, C, 2 \rangle, \langle m_2, C, 2 \rangle
\]
Encodings/Reductions

• MIP solution
  • [makespan] multi-commodity flow model, small graphs, ≤ 50 agents (Yu and LaValle, 2013)

• SAT solution
  • [SIC]: on small maps may eventually beat CBS (Surynek et al, 2016)

• CP solution
  • [makespan] Variables choose a path for each agent, dynamic variable domain (J Wang et al, 2019)

• SMT solution
  • [SIC]: Lazy construction of SAT model, better than CBS on tight maps, small number of agents (Surynek, 2019)

• ASP solution
  • [SIC] a tighter model, uses unsatisfiable core based optimization, small graphs ≤ 16 agents (Gomez et al, 2020)
Encodings/Reductions

• Essential Problem for Naive encodings
  
  • $x_{apt.}$ agent $a$ is at position $p$ at time $t$

  • $O(nVT)$ $n$ is number of agents, $V$ is number of vertices, $T$ is time bound

• Too many variables, even for smallish cases
  
  • $2 \times (2m+5) \times 4$ at least for our trivial example

• Tight $T$ is hard to determine, $O(V^3)$ for sum of costs
Constraint Programming for MAPF
Lazy-CBS
CP for MAPF

• CP is an integrative solving approach
  • we can hide subproblems inside global constraints
• We can essentially reimplement CBS inside a CP solver
  • What the advantage?
• Lazy-CBS: a CP based approach like CBS for solving MAPF
Lazy CBS
Abstract CP MODEL

% MASTER PROBLEM
enum AGENT; % set of agents
enum LOCATION; % set of locations
int: max_time; % maximum time allowed
set of int: TIME = 1..max_time;
% start and end locations
array[AGENT] of LOCATION: start;
array[AGENT] of LOCATION: end;
% cost of each agents path
array[AGENT] of var 0..infinity: c; % cost of each agents path
% which agent is permitted at location l at time t (edges omitted)
array[LOCATION, TIME] of var AGENT: permitted;
constraint forall(a in AGENT)
  (cost_of_path(a,start[a],end[a],permitted,c[a]));
solve minimize sum(c);
CP Path Propagator

• A specialized propagator to update path costs!

• `cost_of_path(a,start[a],end[a],permitted,c)`
  
  • update the lower bound on `c`

  • compute the shortest path `p` from `start[a]` to `end[a]` that satisfies the permitted constraints. e.g. `permitted[p[t],t] = a` for all `t` in path `p`

  • `c \geq length(p)`

• Should update `permitted` to ensure no resource conflicts (ignore for now)

• EXACTLY the same algorithm as the CBS individual agent path finder
Cheating

• Wait a minute!

• Size of the model is still $O(nVT)$
  • $\text{array}[\text{LOCATION, TIME}]$ of $\text{var}$ $\text{AGENT: permitted}$;

• Solution:
  • lazily create the $\text{permitted}$ variables as required

• Bonus
  • $\text{max\_time}$ no longer required

• A permitted variable is only required when two agents use the same location at the same time
• EXACTLY the same as the CBS strategy for adding constraints
WHY? (Reimplement CBS)

• Search is not the same!
  • CBS (best first search)
  • Lazy-CBS (depth first search)
• Learning is possible
  • Avoiding repeated work
CBS Weaknesses

• Two independent conflicts
  • purple/yellow and green/blue
CBS Weaknesses

- Two independent conflicts
  - purple/yellow and green/blue
CBS Weaknesses

- Two independent conflicts
- purple/yellow and green/blue
CBS Weaknesses

• Two independent conflicts
  • purple/yellow and green/blue

CBS has to resolve the same problem in multiple nodes
Lazy CBS Search

- CP solvers are **astoundingly bad** at optimizing sum objectives
  - e.g. Sum of path costs $\sum_{a \in \text{agents}} c_a$
- Core-guided optimization
  - Assume the best possible solution: each path cost $c$ is the shortest
  - Find a contradiction (core)
  - Rewrite the objective with this knowledge
  - Assume the best possible solution under the rewriting
  - **repeat** until solution found
- Core guided optimization requires explanation of failure
Why is a path is too long

- Bound $c_1 \leq 5$. Explain why the path from $s_1$ to $g_1$ is more than 5
- Search backwards from goal at time 5
- Mark as forbidden blocked resources that are reached (pink)
- Search from start at time 0
- Skip over unmarked blocked resources (black), i.e. treat them as available
- Collect reached marked blocked resources $R$ (blue)
- Explanation is $R \rightarrow c_1 > 5$
Lazy CBS in Action

- Initial Problem

- Each Agent constrained to have path cost no greater than shortest path

\[ c_1 : [0, 9] \]
\[ c_2 : [0, 9] \]
Lazy CBS in Action

- Path propagator for agent 1 discovers lower bound on path length
- Generates example path

$c_1 : [9, 9]$  
$c_2 : [0, 9]$
• Similarly for agent 2

• Conflict discovered
• Introduce permitted variable for overloaded resource
Lazy CBS in Action

- On left branch, discover conflict, and derive explanation

The clauses we learned are independent of any other agents!

\[ c_1 : [9, 9] \]
\[ c_2 : [10, 9] \]
\[ \text{permitted}[u, 5] = 1 \]

\[ \langle \text{permitted}[u, 5] \neq 2 \rangle \rightarrow \langle c_2 \geq 10 \rangle \]
The clauses we learned are independent of any other agents!

- Right branch, discover conflict, add explanation

\( c_1 : [10, 9] \)
\( c_2 : [9, 9] \)

\( \text{permitted}[u, 5] = 2 \)

\[ \langle \text{permitted}[u, 5] \neq 2 \rangle \rightarrow \langle c_2 \geq 10 \rangle \]
\[ \langle \text{permitted}[u, 5] \neq 1 \rangle \rightarrow \langle c_1 \geq 10 \rangle \]
The clauses we learned are independent of any other agents!

\[
\begin{align*}
  c_1 &: [10, 9] \\
  c_2 &: [9, 9]
\end{align*}
\]

\[
\begin{align*}
  \langle \text{permitted}[u, 5] \neq 2 \rangle & \rightarrow \langle c_2 \geq 10 \rangle \\
  \langle \text{permitted}[u, 5] \neq 1 \rangle & \rightarrow \langle c_1 \geq 10 \rangle \\
  \langle c_1 \geq 10 \rangle \lor \langle c_2 \geq 10 \rangle
\end{align*}
\]

• Learned nogood is independent of other agents
Lazy CBS in Action

• Lower Bound: 44

• Assumptions
  • $c_1 \leq 11$
  • $c_2 \leq 11$
  • $c_3 \leq 11$
  • $c_4 \leq 11$
Lazy CBS in Action

• Lower Bound: 44

• Assumptions
  • $c_1 \leq 11$
  • $c_2 \leq 11$
  • $c_3 \leq 11$
  • $c_4 \leq 11$

• Nogood $\langle c_1 \geq 12 \rangle \lor \langle c_2 \geq 12 \rangle$
Lazy CBS in Action

- Lower Bound: 45

- Assumptions
  - $c_1 \leq 12$
  - $c_2 \leq 12$
  - $c_3 \leq 11$
  - $c_4 \leq 11$
  - $z_1 \leq 1$
  - $z_1 = \langle c_1 \geq 12 \rangle + \langle c_2 \geq 12 \rangle$
Lazy CBS in Action

- Lower Bound: 45
- Assumptions
  - $c_1 \leq 12$
  - $c_2 \leq 12$
  - $c_3 \leq 11$
  - $c_4 \leq 11$
  - $z_1 \leq 1$
  - $\langle c_3 \geq 12 \rangle \lor \langle c_4 \geq 12 \rangle$
Lazy CBS in Action

- Lower Bound: 46

- Assumptions
  - $c_1 \leq 12$
  - $c_2 \leq 12$
  - $c_3 \leq 12$
  - $c_4 \leq 12$
  - $z_1 \leq 1$
  - $z_2 \leq 1$
  - $z_2 = \langle c_3 \geq 12 \rangle + \langle c_4 \geq 12 \rangle$
Lazy CBS in Action

• Lower Bound: 46

• Assumptions
  • $c_1 \leq 12$
  • $c_2 \leq 12$
  • $c_3 \leq 12$
  • $c_4 \leq 12$
  • $z_1 \leq 1$
  • $z_2 \leq 1$

• Solution Found: Optimal
Independence Detection

- Independence Detection
  - split agents into sets that never interact
  - solve MAPF problems separately
- Problem
  - as number of agents grow
  - nothing is independent
Comparison to CBS

- Similar to Iterative Deepening CBS (IJCAI 2020)
  - Each solve is essentially exploring a whole CBS tree
  - Branches on permitted[\(l, t\)] are analogous to CBS branches
  - **COST** more high-level nodes explored
- **Lazy** introduction of permitted[\(l, t\)]
  - In a leaf node (all existing permitted[\(l, t\)] fixed)
  - New conflict between two agents at location \(l\) time \(t\)
  - Add a new permitted[\(l, t\)] variable, branch on it
- Path propagators = low level search in CBS
- **NEW**: nogoods are universal
  - prevent repeated work across subtrees
  - prevent repeated work in resolves
Lazy CBS Experiments
Lazy-CBS Experiments

Results:
- 20 x 20 grids, 10% obstacles
- 50 instances per size, 5 minute timeout

Gange et al. Lazy CBS 18 / 24
Lazy-CBS Experiments

Results: 20 x 20 grids, 10% obstacles

50 instances per size, 5 minute timeout

Gange et al

Lazy CBS

Results: lak503d

50 instances per size, 5 minute timeout

Gange et al

Lazy CBS
Lazy-CBS Experiments

Results: 20 x 20 grids, 10% obstacles
50 instances per size, 5 minute timeout

Gange et al
Lazy CBS

Results: lak503d
50 instances per size, 5 minute timeout

Gange et al
Lazy CBS

Bonus results: Rectangle symmetry breaking!

---

Number of agents:

<table>
<thead>
<tr>
<th>Number of Agents</th>
<th>CBS</th>
<th>CBSH</th>
<th>ECBS(1.01)</th>
<th>Lazy CBS</th>
<th>Lazy CBS+R</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
</tr>
<tr>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
</tr>
<tr>
<td>40</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
</tr>
<tr>
<td>50</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
</tr>
</tbody>
</table>
Mixed Integer Programming
Approach to MAPF
BCP
Branch and Cut and Price

• A very complicated way of solving a Mixed Integer Program

• Lazily constructing the linear model during solving

• Guaranteed to find optimal solutions

• Allows us to avoid the problem of too big an encoding.
Branch and Cut and Price (roughly)
BCP (roughly)

• Linear Program
  • a set of linear constraints with a linear objective
  • solved by the SIMPLEX algorithm (for our purposes)

• Integer Program
  • (some) variables must take integer variables
  • solved by Branch and Bound

• Branch and Cut and Price
  • Generalisation of Branch-and-Bound
  • Start with small matrix and progressively add rows and columns
Simple Example
Linear Programming
Linear Programming
Branch-and-cut-and-price
Branch-and-cut-and-price
Branch-and-cut-and-price
Branch-and-cut-and-price
Branch-and-cut-and-price
Branch-and-cut-and-price
Branch-and-cut-and-price
BCP Algorithm for MAPF
BCP Algorithm for MAPF

- **Master problem**: select good paths for the agents using LP relaxation
- **Pricing problem**: find better paths for each agent using A*
- **Separation problems**: check for collisions in the selected paths using complete enumeration
Problem Graph

• Define a time-expanded graph:

• A vertex \( v = (c,t) \) is a cell-time pair

• An edge \( e = (v_1,v_2) = ((c_1,t),(c_2,t+1)) \) is a pair of vertices such that:
  • Cell in second vertex is adjacent cell or same cell (wait)
  • Time in second vertex = time in first vertex + 1

• A path is a sequence of vertices such that adjacent vertices are edges
Master Problem

• For each agent $a$, assume a set $P_a$ of all possible paths

• MIP master problem:
  
  • Variable $\lambda_p \in \{0,1\}$ represents proportion of selecting path $p$

  • Path selection constraints: for each agent $a$, select one path from $P_a$

  • Vertex collision constraints: each vertex used by at most one agent

  • Edge collision constraints: each edge and its reverse used by at most one agent
Master Problem

Minimise total cost

subject to

Minimise total cost

Every agent must use (at least) one path

Vertex \( v \) used at most once across all selected paths

Edge \( e \) and its reverse used at most once across all selected paths
• Enough to work with a finite subset $P'_a \subset P_a$

Why use this long path... when this short path exists? (provided it doesn't conflict)
Restricted Master Problem

Minimise total cost

\[ \min \sum_{a \in A} \sum_{p \in P_a'} c_p \lambda_p \]

subject to

\[ \sum_{p \in P_a'} \lambda_p \geq 1 \quad \forall a \in A \]

\[ \lambda_p \geq 0 \quad \forall a \in A, p \in P_a' \]

Every agent must use (at least) one path

Minimise total cost
Separating Vertex Conflicts

- Project to network flow formulation: \( x_v = \sum_{a \in A} \sum_{p \in P_a} x_v^p \lambda_p \)
- Create constraint if \( x_v > 1 \)
Separating Edge Conflicts

- Project to network flow formulation: \( y_e = \sum_{a \in A} \sum_{p \in P_a} y_p \lambda_p \)
- Create constraint if \( y_e + y_{e'} > 1 \)
Pricing Problem

• Use existing A* code with modified objective function:

\[ \min \bar{c}_p := c_p - \alpha_a - \sum_{v \in V} \beta_v x^p_v - \sum_{e \in E} \gamma_e (y^p_e + y_{e'}^p) \]

- Reduced cost of path \( p \)
- Value of dual variable for set cover constraint
- Value of dual variable for vertex collision constraint
- Value of dual variable for edge collision constraint
- Variable = 1 if path \( p \) uses vertex \( v \)
- Variable = 1 if path \( p \) uses edge \( e = (i,j) \)
- Variable = 1 if path \( p \) uses edge \( e' = (i,i) \)

• Path \( p \) may improve the master problem LP solution if \( \bar{c}_p < 0 \) (necessary but not sufficient)
Branching on Vertex

- Solutions have fractional value for $\lambda_p$
- Require branching to ensure they are integer $\{0,1\}$
- Select a vertex $v$ that is fractionally used by an agent $a$
- Agent $a$ must:
  - visit $v$ in one child. (set $\lambda_p = 0$ for paths $p$ where it does not visit)
  - not visit $v$ in other child. (set $\lambda_p = 0$ for paths $p$ where it does visit)
BCP

• BCP iteratively calls three subproblems:

  • **Restricted Master problem:** select good paths using LP relaxation
    
    • Column (variable) represents the proportion of a path selected
    
    • Row (constraint) represents resolving a conflict

  • **Pricing problem:** find better paths for each agent using A*

  • **Separation problems:** check for collisions in the selected paths using complete enumeration

• Master problem selects **fractional proportion** of a path

  • Resolve fractionality by branching on agent-vertex: agent must and must not use vertex (disjoint branching)
BCP

• Implemented in BCP-B algorithm – basic branch-and-cut-and-price

• Now make some improvements:
  • 2 classes of redundant constraints
  • Hierarchical branching rule
Rectangle Symmetry

Rectangle Conflicts

\[
\sum_{p \in P_{a_1}} \sum_{e \in E_1} y_e^p \lambda_p + \sum_{p \in P_{a_2}} \sum_{e \in E_2} y_e^p \lambda_p \leq 3 \quad \forall (a_1, E_1, a_2, E_2)
\]

Fractional Corridor Conflicts
Corridor Conflicts

\[
\sum_{p \in P_{a_1}} y_p^{p_{((l_1,t),(l_2,t+1))}} \lambda_p + \sum_{p \in P_{a_1}} y_p^{p_{((l_1,t+1),(l_2,t+2))}} \lambda_p + \\
\sum_{p \in P_{a_2}} y_p^{p_{((l_2,t),(l_1,t+1))}} \lambda_p + \sum_{p \in P_{a_2}} y_p^{p_{((l_2,t+1),(l_1,t+2))}} \lambda_p \leq 1 \\
\forall (a_1, a_2, l_1, l_2, t)
\]
Branching on Path Length

Agent 1 Selected Paths
(c₁,c₂,c₃,c₄,c₅) – cost 5, proportion 0.5
(c₁,c₂,c₃,c₄,c₅) – cost 6, proportion 0.5

Agent 1 cost = 0.5 × 5 + 0.5 × 6 = 5.5

Agent 1 path length ≤ 5

Agent 1 Selected Paths
(c₁,c₂,c₃,c₄,c₅) – cost 5, proportion 1
(c₁,c₂,c₃,c₄,c₅) – cost 6, proportion 0

Agent 1 cost ≤ 5

Agent 1 path length ≥ 6

Agent 1 Selected Paths
(c₁,c₂,c₃,c₄,c₅) – cost 5, proportion 0.0
(c₁,c₂,c₃,c₄,c₅) – cost 6, proportion 1

Agent 1 cost ≥ 6
BCP

- BCP-B
  - basic branch and cut and price
  - vertex and edge separation
  - pricing problem solved by A*
  - branching on agent uses vertex (location, time) or does not

- BCB
  - add rectangle conflict cuts
  - add corridor conflict cuts
  - first branch on path length of agents
Experiments
Set-up

- Test 4 algorithms on Intel Xeon E5-2660 V3 CPU at 2.6 GHz with 5 min time limit:
  - BCP
  - BCP-B
  - CBSH
  - CBSH-RM – state-of-the-art as of AAAI19 (Jan 2019)

- 1350 standard benchmarks on 4 maps:
Results

- Close all but 15 instances
- Improved to 6 instances in new unpublished work

Figure 3: Success rate for each map and number of agents. Higher is better.
Comparison to CBS

Pros

• Reasons across all conflicts and agents simultaneously

• Immediately recalls all paths generated anywhere in high-level search tree

• Extendable with other types of conflicts

• Exploits advances in MIP
  • Primal heuristics!

• Anytime

• Tight lower bound even if no feasible solution (upper bound) is found

Cons

• Difficult to understand

• Difficult to implement

• 2000 lines of code for A*

• 12000 lines of code for separators and glue to A*

• ??? million lines of code for MIP solver (SCIP)

• ??? million lines of code for LP solver (CPLEX)
Conclusions
Conclusions

• Straight CP solution to MAPF doesn't scale
  • Lazy CBS
    • Benders decomposition (relaxing permitted)
    • Lazy variable addition
    • Core-guided search
• Straight MIP solution to MAPF doesn't scale
  • BCP
    • Dantzig-Wolfe decomposition
    • Specialized Column generator (low level path finding with arbitrary costs)
    • Specialised Cuts (rectangle, corridor, ...)


Discrete Optimization (DO)

• If your problem in NP-hard consider DO techniques

• Straight out of the box may not work

  • But there are lots of DO decomposition approaches
    
    • Benders, Dantzig-Wolfe, and Lagrangian Decomposition (MIP)

    • Logic-Based Benders, Subproblem encapsulation (CP)
MAPF Arms Race

• Methods from path planning (PP) and discrete optimization (DO) crossover

• PP to DO:
  • rectangle symmetries
  • low-level search

• DO to PP
  • corridor symmetries
  • disjoint splitting
  • "depth-first" search
"Better than A*"

- Core-guided search versus A*
  - Use Core-guided search as a plug in replacement for A*
- Features required
  - Combinatorial state space
  - Poor heuristics
  - "repeated subproblems"
The End