Plan Merging Project

Nico Sauerbrei and Paul Raatschen

5. September 2022

Agenda

- Problem Setting
- ASP Components
- Solvers
  - Iterative Solving
  - Prioritized Planning
  - Conflict Based Search
- Benchmarking
- Conclusion

Problem Setting

- MAPF through plan merging.
- Non anonymous MAPF.
- Main focus on satisfiability, i.e. finding a set of valid, conflict free paths for all agents.
- Metrics of interest for valid solutions:
  - Sum of Costs (SoC) := The sum of all time steps required by each agent to reach its target.
  - Makespan := The maximum number of time steps required for all agents to reach their target
- Investigate tradeoff between runtime and solution quality with different approaches.

Conflict Detection

- Paths are represented by position atoms

```
position(R,(X,Y),T)
```

- Conflict types:

```
conflict(vertex,R,R',(X,Y),T) :-
position(R,(X,Y),T), position(R',(X,Y),T), R!=R'.

conflict(edge,R,R',((X,Y),(X',Y')),T-1) :-
position(R,(X,Y),T-1), position(R,(X',Y'),T),
position(R',(X',Y'),T-1), position(R',(X,Y),T), R!=R'.

conflict(vertex,R,R',(X,Y),T) :-
position(R,(X,Y),T), goal(R',(X,Y)),
goalReached(R',T'), R!=R', T >= T'.
```

Single Agent Pathfinding

Adapted Asprilo encoding based on Clingo's incremental mode.

```
{ move(r,D,t) : direction(D) } 1.
position(r,(X+DX,Y+DY),t) :- move(r,(DX,DY),t),
position(r,(X,Y),t-1).
:- move(r,(DX,DY),t), position(r,(X,Y) ,t-1),
not node((X+DX,Y+DY)).
position(r,C,t) :- position(r,C,t-1), not move(r,_,t).
```

Check for each iteration if the goal has been reached.

```
#program check(t).
#external query(t).
goalReached(r,t) :- goal(r,C), position(r,C,t), not block(t)
:- not goalReached(r,t), query(t).
```

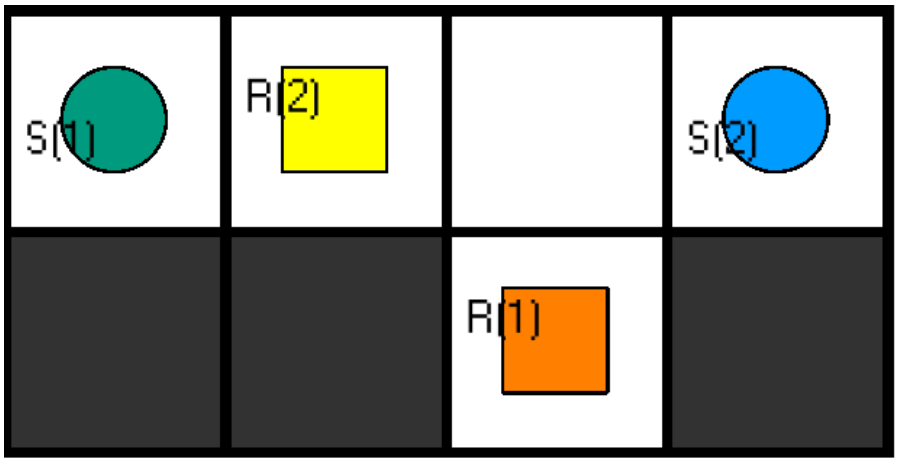
Iterative Solving

Idea:

- Solving one conflict at a time
- Seperating conflict based on complexity
  - Conflict which can be solved by one agent waiting
  - Conflicts which can be solved by one agent evading the other

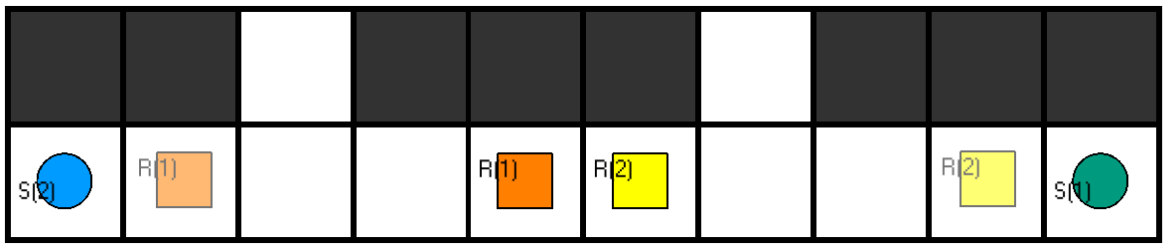
Vertex Conflict Handling

- Precondition: Conflict was found which can be solved with one agent waiting
- Agent which has to wait gets chosen based on priority:
  - Priority 1: The path of the agent is shorter than the one of the other agent
  - Priority 2: The agent has to move, for the other agent to continue



Edge Conflict Handling

- From the point of conflict both agents restep their own path to find a suitable point to evade the other agent
- Once all points are found the one nearest to the conflict time gets choosen
- The agent who dodges, evades onto the spot before the conflict would occur, waiting there before continuing as normal



Iterative Solving

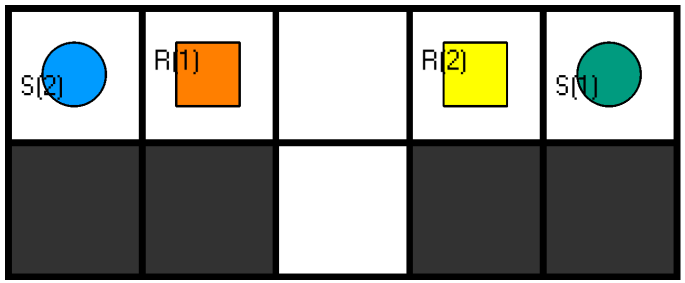
Algorithm 1: Iterative solving

```
current.plans ← Node with optimal paths for all agents
while True do
    conflict ← Conflicts between paths p_1, ..., p_n
    if no conflicts in current.plans then
        return current.plans
    else
        if edge conflict in conflicts then
            solveEdge()
            continue
        else
            solveVertex()
        end
    end
end
```

Prioritized Planning

- Idea: Plan agents paths sequentially according to a priority order.
- Agent have to avoid conflicts with agents of a higher priority.
- Prioritized Planning is not a complete solver.
- No guarantees on the quality of the solution.
- Runtime linear in the number of agents.

Unsolvable instance for PP



Prioritized Planning Algorithm

Algorithm 2: Prioritized Planning

order $\leftarrow \{a_1, ..., a_n\}$
plans $\leftarrow \emptyset$
for *agent* $a_i \in$ *order* **do**
  `// Plans conflict free path for` a_i
  $plan_i \leftarrow$ plan_path(a_i,plans)
  if $plan_i.cost < \infty$ **then**
    plans$\leftarrow$ plans $\cup$ $plan_i$
  else
    return $\emptyset$
  end
end
return *plans*

Avoiding conflicts

Avoiding vertex and edge conflicts with known plans.

```
:- position(r,(X,Y),t), position(R,(X,Y),t), r!=R.

:- position(r,(X,Y),t-1), move(r,(DX,DY),t),
position(R,(X+DX,Y+DY),t-1), position(R,(X,Y),t), r!=R.

:- position(r,(X,Y),t), goal(R,(X,Y)), goalReached(R,T),
r!=R, t > T.

block(1..T-1) :- position(R,C,T), goal(r,C), r!=R.
```

Schedule Optimization

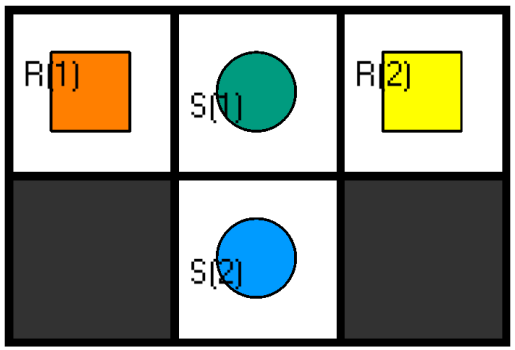
- Initial order of agents impacts solvability and solution quality.
- **Problem**: Determine optimal order without actually testing every permutation.
- Heuristic approach: Order agents according to the number of conflicts in their initial plans.
- Idea: Agents with few conflicts should be prioritized, since they are less likely to block following agents.
- No formal guarantee on improving the solution $\rightarrow$
  Has to be evaluated empirically.

Backtracking

- Idea: Reorder agents if the current order is unsolvable.

Algorithm 3: Backtracking

order $\leftarrow \{a_1, ..., a_n\}$
orderings $\leftarrow \{$order$\}$
while *maxIter* > 0 **do**
  if *prioritized_planning(order) not satisfiable* **then**
    orderings $\leftarrow$ orderings $\cup \{$All orders beginning with $a_1, ..., a_i\}$
    Change agents a_i and a_{i-1}
    order $\leftarrow \{a_{i-1}, ..., a_n\}$
    if *order* $\in$ *orderings* **then**
      order $\leftarrow$ Order with maximum distance to known orders.
    end
    orderings $\leftarrow$ orderings $\cup \{$*order*$\}$
    maxIter $\leftarrow$ maxIter - 1
  else
    return *prioritized_planning(order)*
  end
end



- Increases runtime complexity to $N!$ in the worst case.
- Marking unsolvable orders should improve performance on average.

Conflict Based Search

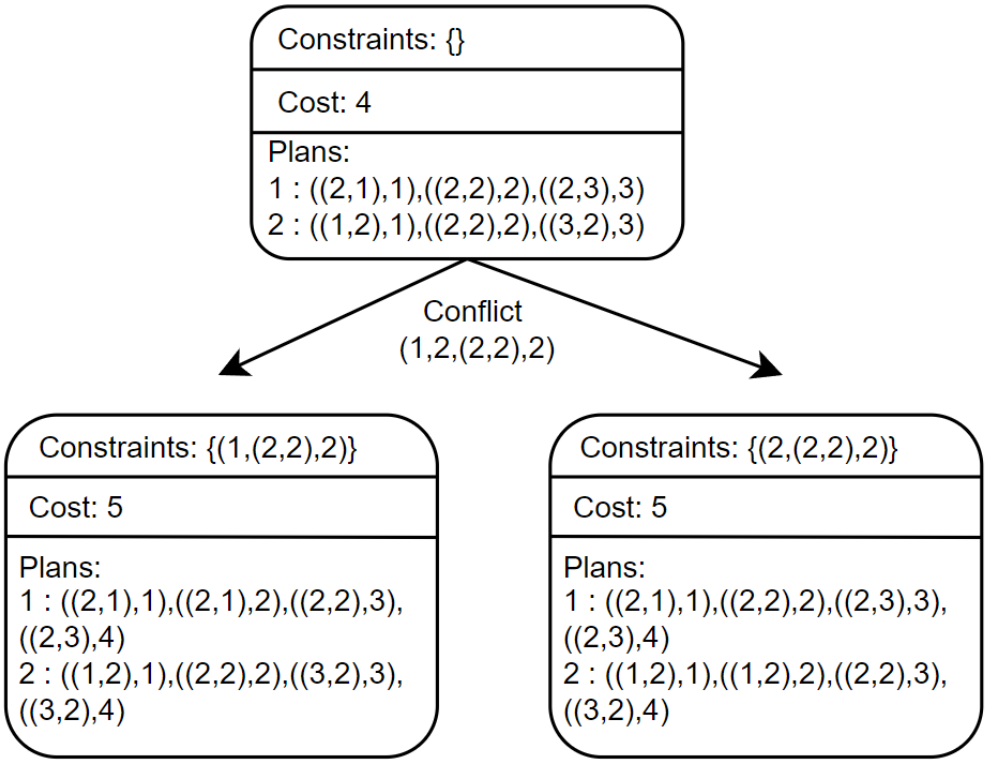
- Complete and optimal MAPF algorithm by Sharon et al.[1]
- Two level search.
  - High level search of conflict tree.
  - Low level path finding search.
- Our implementation: High level search in Python, low level search and conflict detection in ASP.

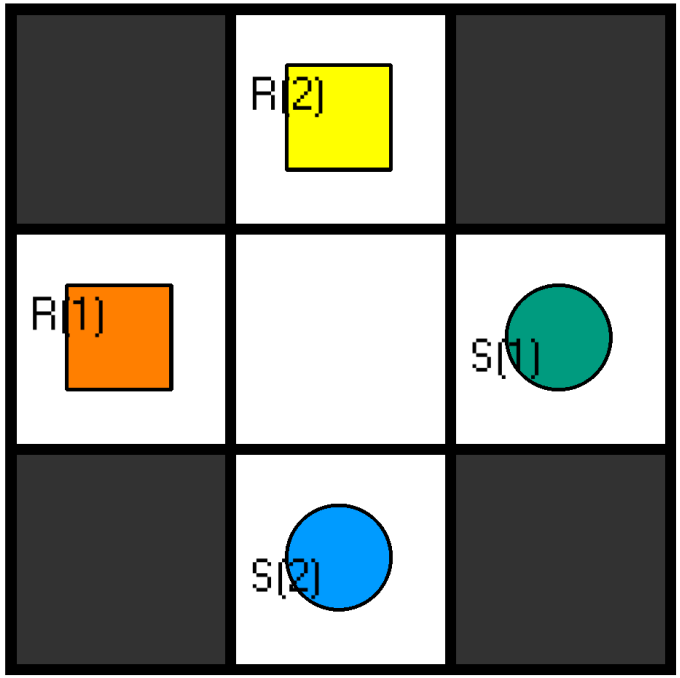
CBS Algorithm

Algorithm 4: Conflict Based Search (Base)

root $\leftarrow$ Node with optimal paths for all agents
open $\leftarrow$ {root}
while *open not empty* **do**
  current $\leftarrow$ lowest cost node from open
  if no conflicts in current.plans **then**
    return current.plans
  else
    conflict $\leftarrow$ conflict between agents a_1, a_2 at (x, y, t)
    for *agent* a_i *in conflict* **do**
      node $\leftarrow$ current with additional constraint (a_i, x, y, t)
      `// computes path for` a_i
      node.low_level(a_i)
      if *node.cost* $< \infty$ **then**
        open $\leftarrow$ open$\cup${node}
      end
    end
  end
end

CBS Example


Constraints: {}
Cost: 4
Plans:
1 : ((2,1),1),((2,2),2),((2,3),3)
2 : ((1,2),1),((2,2),2),((3,2),3)
Conflict
(1,2,(2,2),2)
Constraints: {(1,(2,2),2)}
Cost: 5
Plans:
1 : ((2,1),1),((2,1),2),((2,2),3),
((2,3),4)
2 : ((1,2),1),((2,2),2),((3,2),3),
((3,2),4)
Constraints: {(2,(2,2),2)}
Cost: 5
Plans:
1 : ((2,1),1),((2,2),2),((2,3),3),
((2,3),4)
2 : ((1,2),1),((1,2),2),((2,2),3),
((3,2),4)


R(2)
R(1)
S(1)
S(2)

Constraints

Constraining agent positions and movements.

```
:- constraint(r,C,t), position(r,C,t).

:- constraint(r,C,M,t-1), position(r,C,t-1), move(r,M,t).

block(1..T-1) :- constraint(r,C,T), goal(r,C).
```

Cost Functions for CBS

- CBS can optimize different cost functions.
- Cost function determines the value used to sort nodes in the open queue.
  - Sum of costs.
  - Makespan
  - Greedy := Cost function + number of conflicts in the node.
- Greedy search can produce sub-optimal solutions, but should improve performance.

Conflict Bypassing

- Improvement to CBS suggested by Boyarsky et al. [2]
- Bypass conflicts instead of branching if possible.

Algorithm 5: Bypass

conflict ← conflict between agents a_1, a_2 at time t
for *agent* a_i *in conflict* **do**
  node ← current with additional constraint for a_i at time t
  node.low_level(a_i)
  if *node.cost* $\leq$ *current.cost and node.ccount* $<$ *current.ccount*
   then
    open ← open$\cup\{$node$\}$
    break
  end
end

- Reduces number of branches in conflict tree.

Meta Agent CBS

- Idea: Merge heavily conflicting agents into meta agents.
- Trade of between high level and low level search.
- Requires tracking of conflicts through conflict matrix and MAPF solver for low level search.

Algorithm 6: Meta Agent CBS

conflict $\leftarrow$ conflict between agents a_1, a_2 at time t
conflict_matrix(a_1,a_2) $\leftarrow$ conflict_matrix(a_1,a_2) + 1
if *conflict_matrix(a_1,a_2)* $>$ *merge threshold* **then**
    merge(a_1,a_2)
    `// computes path for meta agent` $a_{(1,2)}$
    node.low_level($a_{(1,2)}$)
    if *node.cost* $< \infty$ **then**
        open $\leftarrow$ open$\cup${node}
    end
end

Meta Agent CBS II

Adapted single agent pathfinding ASP encoding as a low level MAPF solver. Main difference:

```
cost(N,t) :-
N = #sum{1, R, T : position(R,C,T), not goal (R,C)}.

#minimize{ N : cost(N,t)}.
```

Problem: We struggled to find an efficient minimization statement. Current implementation only provides close to optimal solutions in terms of makespan.

Improved CBS

- Further improvements to CBS suggested by Boyarsky et al. [3]

Improved CBS

- Further improvements to CBS suggested by Boyarsky et al. [3]
- 1. Merge Reset: Whenever two (meta) agents are merged to a meta agent, restart the search from scratch with the new meta agent already merged.

Improved CBS

- Further improvements to CBS suggested by Boyarsky et al. [3]
- 1. Merge Reset: Whenever two (meta) agents are merged to a meta agent, restart the search from scratch with the new meta agent already merged.
- 2. Prioritizing conflicts

Improved CBS

- Further improvements to CBS suggested by Boyarsky et al. [3]
- 1. Merge Reset: Whenever two (meta) agents are merged to a meta agent, restart the search from scratch with the new meta agent already merged.
- 2. Prioritizing conflicts
  - Cardinal conflict := Both constraints lead to an increase in cost.

Improved CBS

- Further improvements to CBS suggested by Boyarsky et al. [3]
- 1. Merge Reset: Whenever two (meta) agents are merged to a meta agent, restart the search from scratch with the new meta agent already merged.
- 2. Prioritizing conflicts
  - Cardinal conflict := Both constraints lead to an increase in cost.
  - Semi cardinal conflict := One constraint leads to an increase in cost.

Improved CBS

- Further improvements to CBS suggested by Boyarsky et al. [3]
- 1. Merge Reset: Whenever two (meta) agents are merged to a meta agent, restart the search from scratch with the new meta agent already merged.
- 2. Prioritizing conflicts
  - Cardinal conflict := Both constraints lead to an increase in cost.
  - Semi cardinal conflict := One constraint leads to an increase in cost.
  - Non cardinal conflict := Both constraints do not increase cost.

Improved CBS

- Further improvements to CBS suggested by Boyarsky et al. [3]
- 1. Merge Reset: Whenever two (meta) agents are merged to a meta agent, restart the search from scratch with the new meta agent already merged.
- 2. Prioritizing conflicts
  - Cardinal conflict := Both constraints lead to an increase in cost.
  - Semi cardinal conflict := One constraint leads to an increase in cost.
  - Non cardinal conflict := Both constraints do not increase cost.
- Evaluate conflicts in a node and branch in order: cardinal $>$ semi cardinal $>$ non cardinal.

Improved CBS

- Further improvements to CBS suggested by Boyarsky et al. [3]
- 1. Merge Reset: Whenever two (meta) agents are merged to a meta agent, restart the search from scratch with the new meta agent already merged.
- 2. Prioritizing conflicts
  - Cardinal conflict := Both constraints lead to an increase in cost.
  - Semi cardinal conflict := One constraint leads to an increase in cost.
  - Non cardinal conflict := Both constraints do not increase cost.
- Evaluate conflicts in a node and branch in order: cardinal $>$ semi cardinal $>$ non cardinal.
- Prioritizing cardinal conflicts should reduce size of the conflict tree. **Problem**: Evaluating conflicts is costly in our implementation.

Bechmarking - CBS

Table with maxed out execution time

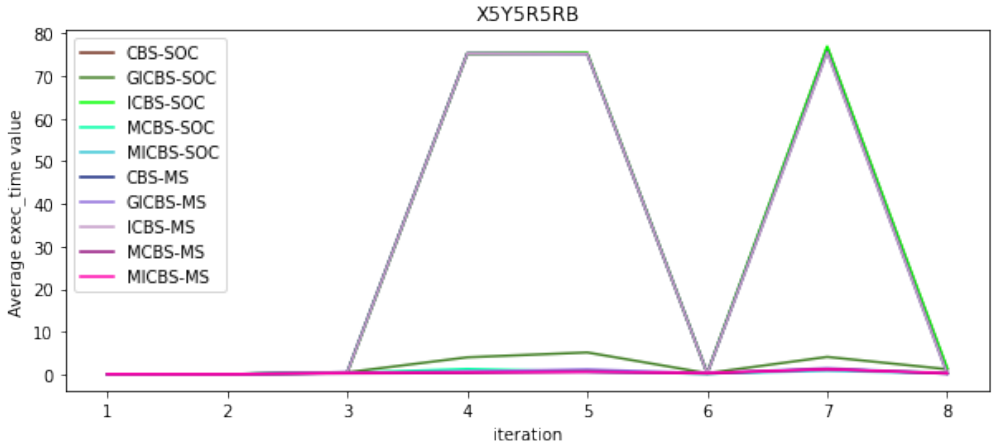
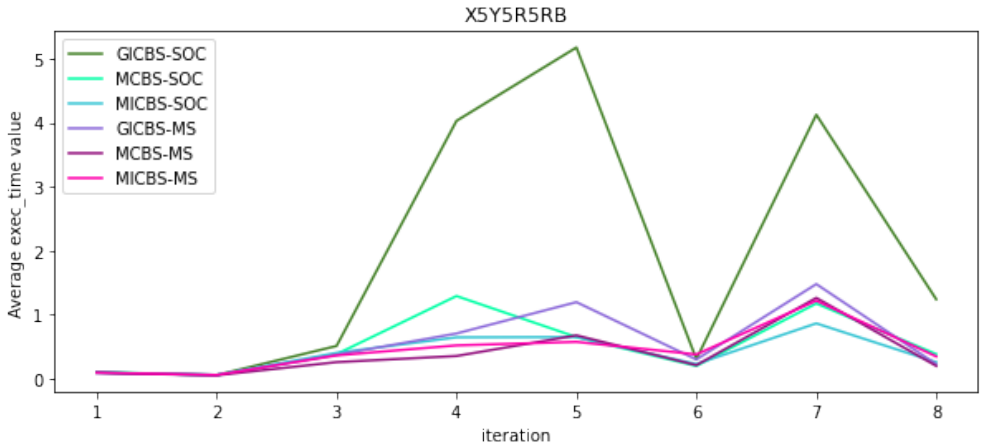


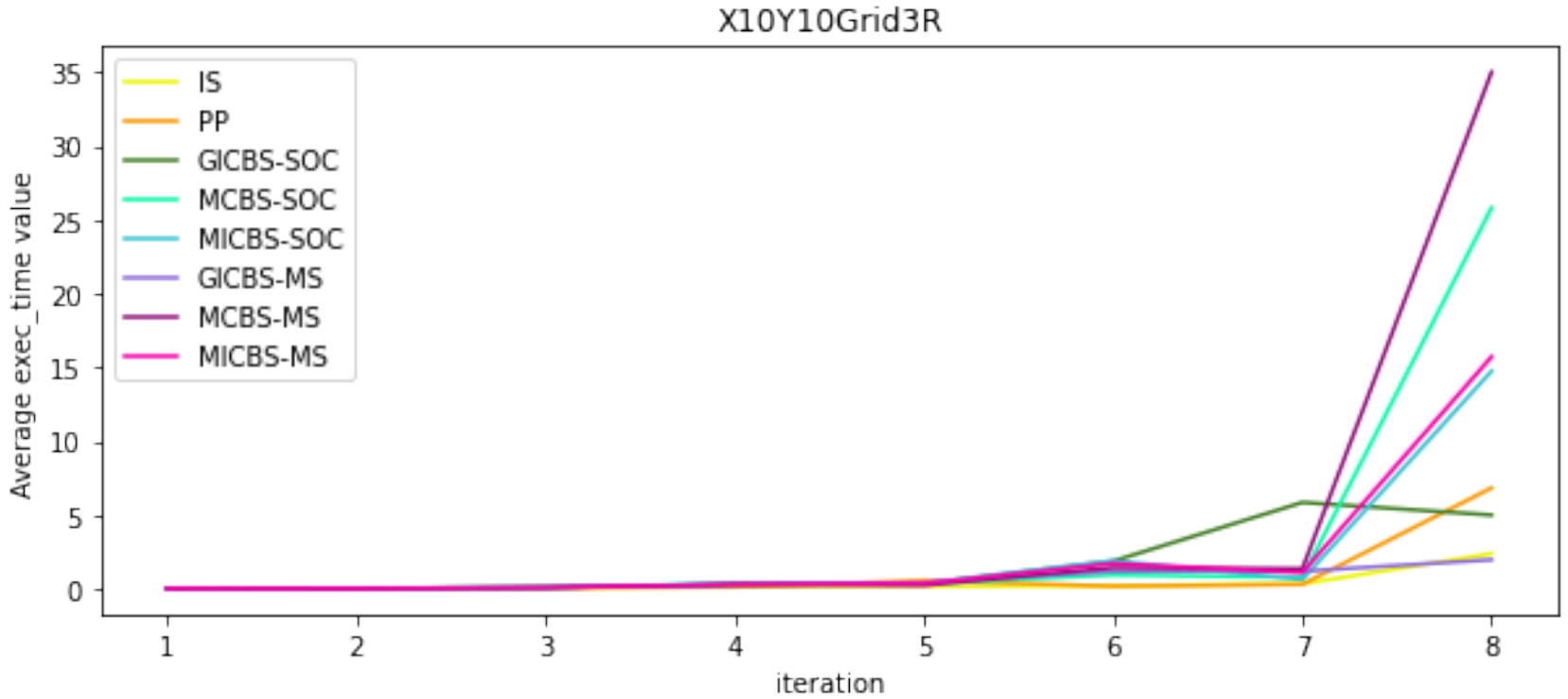
Table with slower CBS-versions removed



- We benchmarked our solvers based on instances with ever increasing difficulty.
- Base instance was the sze of 5 x 5 and had 5 agent, with an extra random wall getting placed per iteration.
- One iteration contains 4 different instances from which we took the average.

Benchmarks - Compared

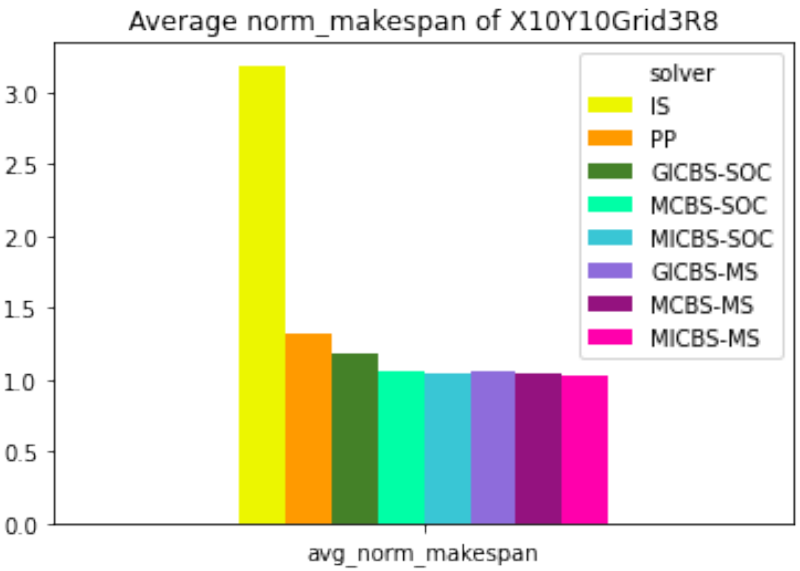
10 by 10 instance, with 3 vertical and horizontal lines, with one door between each section.

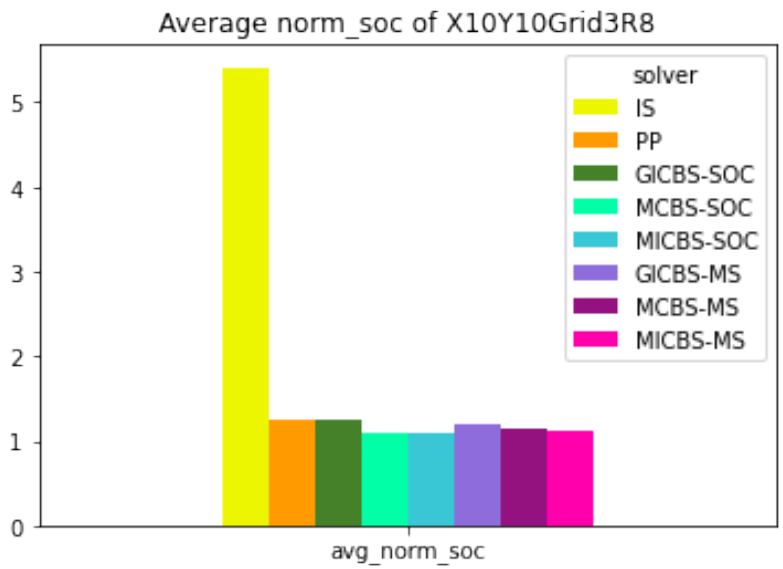


Iteration = number of Agents.

Benchmarks - Makespan / SOC

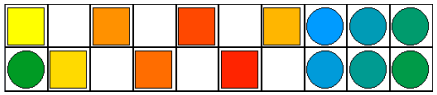
- Makespan and Sum of Cost in relation to the unsolved instance





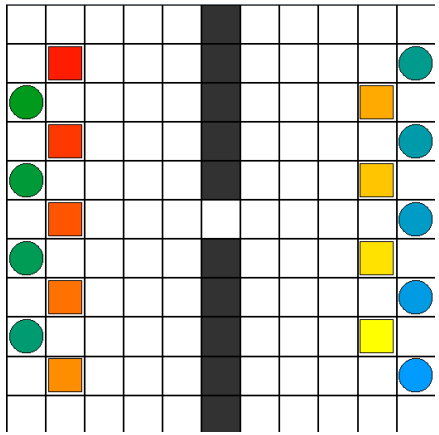
Benchmarks - Sauerbrei Raatschen

Plan-Sauerbrei-Raatschen-2



Solver	Ms	SoC	NoM	ET
IS	26	176	50	0.470
PP	8	49	46	0.122
PP-OPT	12	45	44	0.178
CBS-SOC	12	45	44	11.591
GICBS-SOC	12	45	44	3.603
ICBS-SOC	12	45	44	11.843
MCBS-SOC	8	47	42	2.683
MICBS-SOC	8	53	44	2.070
CBS-MS	8	50	46	9.09
GICBS-MS	11	51	44	13.544
ICBS-MS	8	50	44	27.841
MCBS-MS	8	48	44	2.017
MICBS-MS	8	53	44	2.016

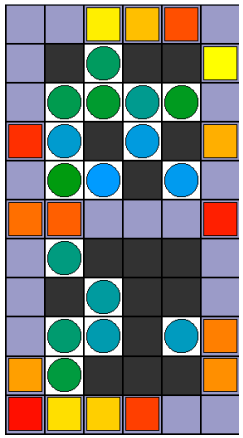
Plan-Sauerbrei-Raatschen-1



Solver	Ms	SoC	NoM	ET
IS	31	279	133	2.088
PP	23	153	129	1.347
PP-OPT	28	171	135	2.355
MICBS-SOC	21	153	147	788.276

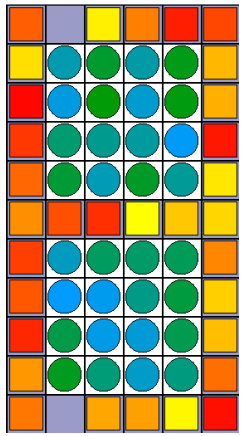
Benchmarks - Steven Pan

x6_y11_r16_fo1_cmw1_cx4_cy4_rom2
_instance_002



Solver	Ms	SoC	NoM	ET
PP	20	201	160	38.000
PP-OPT	23	229	175	44.000
GICBS-MS	14	175	140	141.726

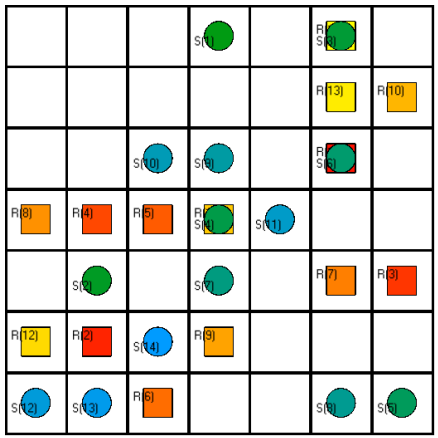
x6_y11_r32_fo2_cmw1_cx4_cy4_rom2
_instance_004



Solver	Ms	SoC	NoM	ET
PP	22	353	285	12.500
PP-OPT	24	339	291	1.480

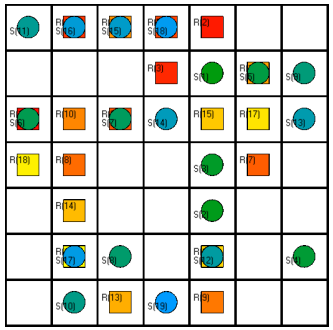
Benchmarks - Moek Andreev

ex14



Solver	Ms	SoC	NoM	ET
PP	10	73	71	0.471
PP-OPT	10	69	69	0.728
CBS-SOC	9	65	63	2.848
GICBS-SOC	9	69	65	4.190
ICBS-SOC	9	65	63	2.222
MCBS-SOC	9	65	63	2.658
MICBS-SOC	9	65	63	113.399
CBS-MS	9	71	67	2.578
GICBS-MS	9	72	63	2.561
ICBS-MS	9	72	63	3.743

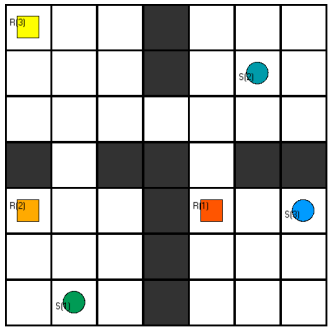
ex8



Solver	Ms	SoC	NoM	ET
PP	10	98	96	0.483
PP-OPT	10	108	100	0.903
CBS-SOC	10	94	94	5.806
GICBS-SOC	10	99	96	4.684
ICBS-SOC	10	94	92	3.762
MCBS-SOC	10	95	94	7.440
CBS-MS	10	101	100	4.535
GICBS-MS	10	100	96	3.805
ICBS-MS	10	100	96	5.467

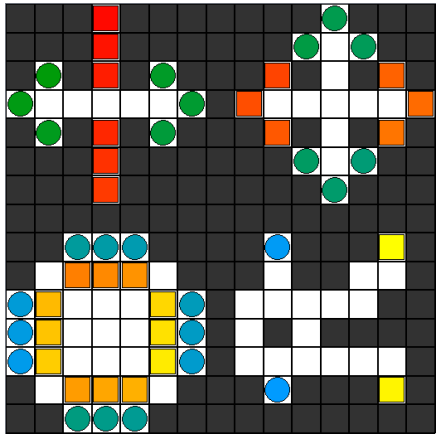
Benchmarks - Jan Behrens

small_room



Solver	Ms	SoC	NoM	ET
IS	14	40	29	0.155
PP	11	30	29	0.100
PP-OPT	11	30	29	0.188
CBS-SOC	11	30	29	0.762
GICBS-SOC	11	30	29	0.376
ICBS-SOC	11	30	29	0.764
MCBS-SOC	11	30	29	0.492
MICBS-SOC	12	33	31	0.639
CBS-MS	11	30	29	0.784
GICBS-MS	11	30	29	0.357
ICBS-MS	11	30	29	0.768
MCBS-MS	11	30	29	0.410
MICBS-MS	12	33	31	0.634

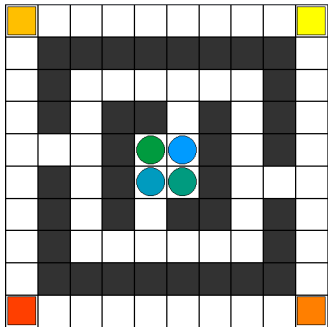
mix



Solver	Ms	SoC	NoM	ET
PP	11	195	162	0.597
PP-OPT	11	195	162	0.886
MICBS-SOC	11	185	164	53.831
MICBS-MS	11	185	164	27.951

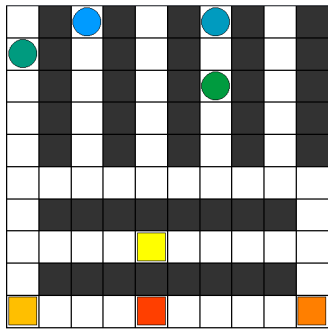
Benchmarks - Glätzer Akil

mini_maze



Solver	Ms	SoC	NoM	ET
IS	23	90	64	0.443
PP	15	58	56	0.234
PP-OPT	15	58	56	0.391
CBS-SOC	15	58	56	0.376
GICBS-SOC	15	58	56	0.361
ICBS-SOC	15	58	56	0.398
MCBS-SOC	15	58	56	0.364
MICBS-SOC	15	58	56	0.372
CBS-MS	15	58	56	0.408
GICBS-MS	15	58	56	0.401
ICBS-MS	15	58	56	0.358
MCBS-MS	15	58	56	0.453
MICBS-MS	15	58	56	0.494

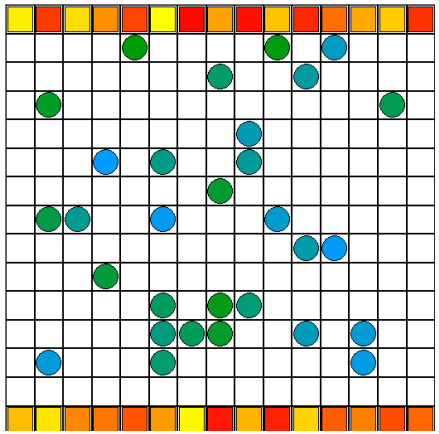
warehouse



PP	17	60	60	0.255
PP-OPT	17	60	60	0.443
CBS-SOC	17	60	60	0.366
GICBS-SOC	17	60	60	0.377
ICBS-SOC	17	60	60	0.383
MCBS-SOC	17	60	60	0.370
MICBS-SOC	17	60	60	0.353
CBS-MS	17	60	60	0.384
GICBS-MS	17	60	60	0.380
ICBS-MS	17	60	60	0.364
MCBS-MS	17	60	60	0.370
MICBS-MS	17	60	60	0.363

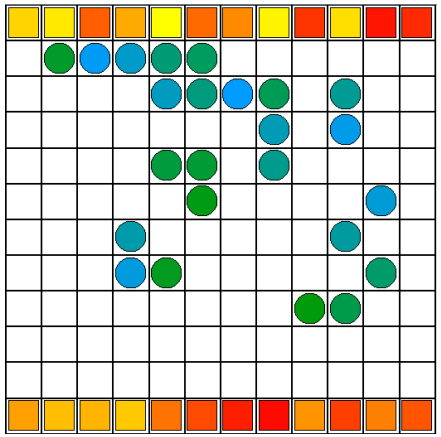
Benchmarks - Cordova Khatova

30_merged_plans



Solver	Ms	SoC	NoM	ET
PP	19	316	307	2.382
PP-OPT	19	321	307	4.635
CBS-SOC	19	303	303	9.287
GICBS-SOC	19	304	303	12.997
ICBS-SOC	19	303	303	10.856
MCBS-SOC	19	303	303	11.494
MICBS-SOC	19	303	303	61.630
CBS-MS	19	303	303	15.251
GICBS-MS	19	337	307	17.904
ICBS-MS	19	337	307	22.749
MCBS-MS	19	303	303	14.711

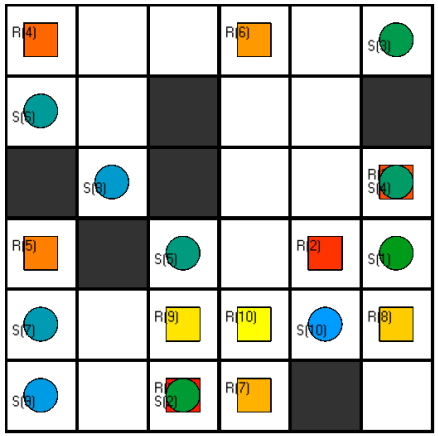
24_merged_plans



Solver	Ms	SoC	NoM	ET
PP	15	263	239	1.374
PP-OPT	15	261	237	2.384
CBS-SOC	16	236	235	216.957
GICBS-SOC	15	249	233	37.274
ICBS-SOC	16	236	233	265.565
CBS-MS	15	260	237	30.610
GICBS-MS	15	258	233	22.595
ICBS-MS	15	259	239	57.423

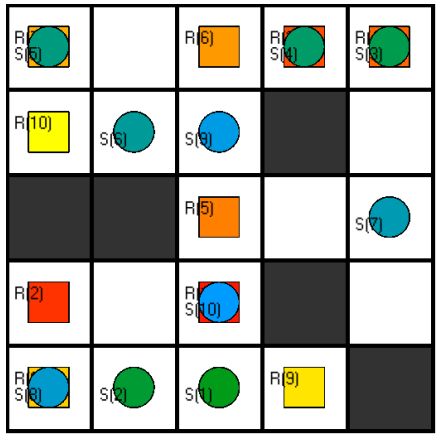
Benchmarks - Nemes Murphy

6_by_6_agents_10_obstacles_6_1



Solver	Ms	SoC	NoM	ET
PP	10	58	56	0.149
PP-OPT	10	53	52	0.282
CBS-SOC	10	50	48	2.363
GICBS-SOC	10	52	50	0.821
ICBS-SOC	10	50	48	2.372
MCBS-SOC	10	51	50	0.732
MICBS-SOC	10	51	48	2.387
CBS-MS	10	65	56	20.891
GICBS-MS	10	52	50	1.199
ICBS-MS	10	52	50	1.879
MCBS-MS	10	54	50	13.019

5_by_5_agents_10_obstacles_5_2



Solver	Ms	SoC	NoM	ET
PP	9	52	41	4.048
MCBS-SOC	9	52	47	13.441
MICBS-SOC	9	46	43	32.535
MCBS-MS	9	52	47	10.904
MICBS-MS	9	46	43	28.284

Conclusion

- **Iterative Solving**: Poor range of solvable instances and solution quality, decent runtime.
- **Prioritized Planning**: Acceptable runtime and solution quality, backtracking can be further improved.
- **CBS**: Impractical runtime for larger instances, ICBS, MACBS and GCBS are not well optimized $\rightarrow$ lots of room for improvement.
- To many solvers. Focus on 1-2 solvers would have left more time for optimization.

Sources I

[1] Guni Sharon **andothers**. “Conflict-based search for optimal multi-agent pathfinding”. **in***Artificial Intelligence*: 219 (2015), **pages** 40–66.

[2] Eli Boyrasky **andothers**. “Don’t split, try to work it out: Bypassing conflicts in multi-agent pathfinding”. **in***Proceedings of the International Conference on Automated Planning and Scheduling*: **volume** 25. 2015, **pages** 47–51.

[3] Eli Boyarski **andothers**. “ICBS: Improved conflict-based search algorithm for multi-agent pathfinding”. **in***Twenty-Fourth International Joint Conference on Artificial Intelligence*: 2015.

[4] Michal Čáp **andothers**. “Prioritized planning algorithms for trajectory coordination of multiple mobile robots”. **in***IEEE transactions on automation science and engineering*: 12.3 (2015), **pages** 835–849.

Sources II

[5] Jur P Van Den Berg **and** Mark H Overmars. “Prioritized motion planning for multiple robots”. **in***2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*: IEEE. 2005, **pages** 430–435.

[6] Roni Stern **andothers**. “Multi-agent pathfinding: Definitions, variants, and benchmarks”. **in***Twelfth Annual Symposium on Combinatorial Search*: 2019.

[7] Martin Gebser **andothers**. “Experimenting with robotic intra-logistics domains”. **in***Theory and Practice of Logic Programming*: 18.3-4 (2018), **pages** 502–519.

[8] Max Barer **andothers**. “Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem”. **in***Seventh Annual Symposium on Combinatorial Search*: 2014.

Sources III

[9] Nico Sauerbrei **and** Paul Raatschen. *Plan Merging Project Repository*. URL: https://github.com/PaulRaatschen/Plan-Merging-Project-Sauerbrei-Raatschen.

[10] Martin Gebser **andothers**. “Multi-shot ASP solving with clingo”. **in** *Theory and Practice of Logic Programming*: 19.1 (2019), **pages** 27–82.