Satisfying Cooperative Path-Finding

Student: Margarida Ferreira
margaridaacferreira@tecnico.ulisboa.pt

Tutor: Inês Lynce
ines.lynce@tecnico.ulisboa.pt

Novos Talentos em Inteligência Artificial
Fundação Calouste Gulbenkian

September 2018

Abstract

In this project, we studied and evaluated several different techniques to find an optimal solution for the problem of Cooperative Path-Finding (CPF) based on Propositional Satisfiability (SAT). Given a set of agents, each with a start and a goal positions, the task is to find non-colliding paths for the different agents in the least possible number of time steps. CPF is used to model many real-world present-day problems in a wide variety of areas.

Keywords — Cooperative path-finding (CPF), propositional satisfiability (SAT), search methods, sequential increasing search, binary search, independence detection (ID)

1 Introduction

Cooperative Path-Finding (CPF), also know as Multi-Agent Path-Finding (MAPF), Multi-Robot Path Planning (MRPP) or Pebble Motion on a Graph (PMG), has recently received a lot of attention from the Artificial Intelligence (AI) community, owing it not only to its numerous practical and present-day applications but also to the challenges it offers. CPF can be used to model many real-world problems across a wide variety of fields. Some examples include the management of autonomous aircraft towing vehicles [5], autonomous warehouse systems [19], coordinated office robots [18] and traffic optimisation [2]. Apart from its many practical applications, research on optimal solutions for CPF helps shedding light on the theoretical hardness of this problem and others of similar complexity.

Contemporary approaches to solve the CPF problem include polynomial time suboptimal algorithms [16], as well as methods that generate an optimal solution. Optimal solvers used for CPF are usually search-based solvers [8, 7, 3, 9] or logic-based solvers [6, 12, 13]. In this project, we focused exclusively on optimal solving based on propositional satisfiability (SAT).

There has been enough work on SAT-based methods to provide evidence that it produces good results for this problem, as well as for others with which it shares significant characteristics. With this in mind, throughout the duration of this project, we

explored the work that had been done in the area, and studied many of the different SAT-based solutions proposed for this problem.

1.1 The problem

CPF represents an abstraction for a variety of problems in which the task is to relocate a set of physical agents. Each agent is given its initial position in a certain environment and its task is to reach a given goal position, without colliding with any other agent. A collision occurs when two agents are planned to occupy the same vertex at the same time step, or when two agents swap positions over two time steps. At any given time, an agent may either move to an empty adjacent location, or stand still in its current location.

Figure 1 is a graphic representation of a set of agents (coloured triangles) positioned in an environment, and their respective goal positions (circles).

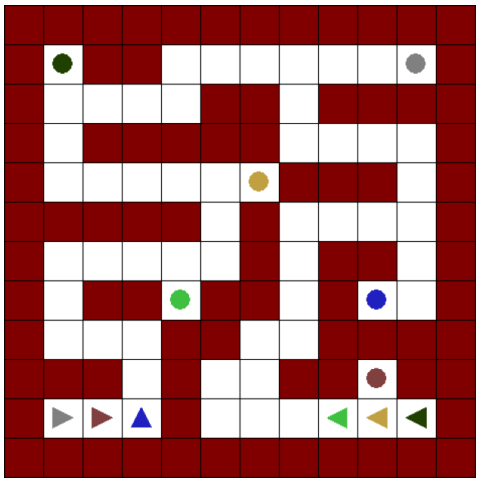


Figure 1: An instance of CPF.

A solution for CPF is then no more than a set of paths, one for each agent, each of which starts at the initial position and ends at their goal. We usually aim at relocating all the agents in the minimum time possible. In other words, we consider a solution optimal if it takes the least possible amount of time for the last agent to reach its goal position.

The complexity of CPF comes from the cooperation required among the agents, in order to comply with the restriction that they must not collide with one another. The more agents are present in the environment, the more interactions there may be between them, thus providing additional complexity to the task of finding a solution. In fact, finding an optimal solution to CPF was shown to be NP-hard [11, 20] as the state-space grows exponentially with the number of agents.

2 Formal definition – Graph modelling

The environment in which the agents move may be modelled using an undirected graph $G = (V, E)$, where $V = \{v_1, v_2, ..., v_n\}$ is a finite set of vertices, each of them representing a possible position for the agents to occupy, and $E \subseteq \binom{V}{2}$ is a set of edges along which the agents may move.

A distribution of the agents in the environment may then be represented by assigning each agent a vertex. Considering $A = \{a_1, a_2, ..., a_\mu\}$ a finite set of agents, an arrangement of the agents in the vertices of G can be fully described by a location function $\alpha : A \to V$. This way, $v_x = \alpha(a_y)$ is interpreted as "agent a_y is located on vertex v_x".

Figure 2 shows the graph (on the right) resulting from a set of three agents positioned in a 4-connected grid environment (on the left).

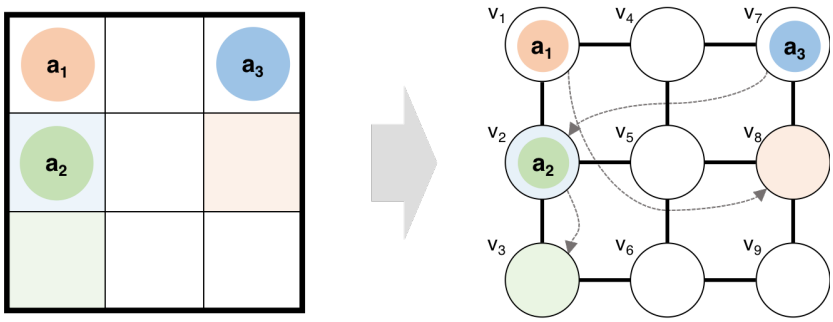


Figure 2: Translation of an instance of CPF into an undirected graph.

Since no two agents may ever occupy the same vertex, α is an injective function, and allows an inverse function $\alpha^{-1} : V \to A$. Then $a_y = \alpha^{-1}(v_x)$ means vertex v_x is occupied by agent a_y, and $\alpha^{-1}(v_x) = \perp$ means no agent is located at vertex v_x.

For an instance of CPF to be completely defined, two particular location functions must be specified: α_0 and α^+, which define the initial and goal arrangements of the agents, respectively.

An example is presented on Figure 3. The initial configuration of the agents (the same represented in Figure 2 on the left, and their goal positions on the right.

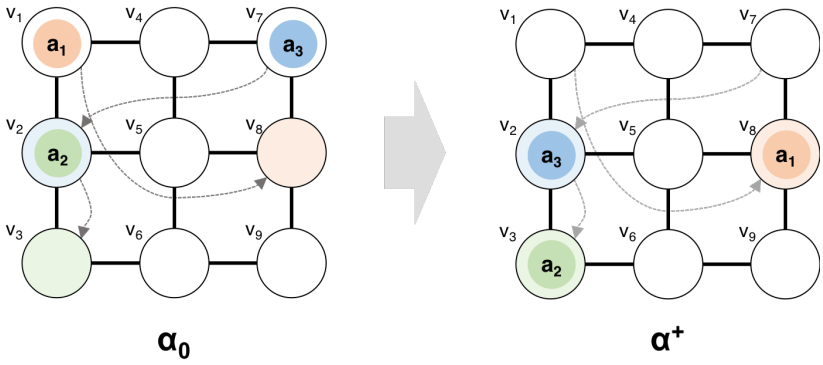


Figure 3: Example of a specification of α_0 and α^+.

An instance of CPF is then fully described as a quadruple:

$$\Sigma = [G = (V, E), A, \alpha_0, \alpha^+]$$

To solve any instance of CPF, we divide time into discrete time steps, and generate a new arrangement for the agents based on their current positions. The validity of the agents' moves in each time step can be dictated by 3 constraints over the graph's elements:

(1) $\forall a \in A$,
either $\alpha_i(a) = \alpha_{i+1}(a)$
or $\{\alpha_i(a), \alpha_{i+1}(a)\} \in E$ holds

(agents move either along an edge, or not at all)

(2) $\forall a \in A, \quad \alpha_i(a) \neq \alpha_{i+1}(a) \Rightarrow \alpha_i^{-1}(\alpha_{i+1}(a)) = \bot$

(agents move to vacant vertices only)

(3) $\forall a, b \in A, \quad a \neq b \Rightarrow \alpha_{i+1}(a) \neq \alpha_{i+1}(b)$

(no two agents may enter the same vertex at the same time)

3 SAT-based Optimal Solving

In this section we provide a brief description of techniques and encodings involved in SAT-based optimal solving for CPF, namely the SIMPLIFIED encoding, two different search techniques, sequential increasing and binary, and the independence detection method.

3.1 SIMPLIFIED encoding

In order to solve CPF using propositional satisfiability, there has to be an encoding from a CPF instance Σ into a propositional formula $F(\Sigma, \eta)$, such that the formula is satisfiable if and only if the CPF instance has a solution for makespan η. Once such formula has been constructed, it is possible to obtain the optimal makespan by querying a SAT-Solver with $F(\Sigma, \eta)$ for varying values of η.

The SIMPLIFIED encoding was first proposed by P. Surynek in 2014 [13], and it has been considered to have the best performance among all CPF encodings [14]. It uses propositional variables $\mathcal{X}^i_{j,k}$ and ε^i_j for each vertex v_j, agent a_k and time step i. $\mathcal{X}^i_{j,k}$ is assigned TRUE if and only if agent a_k is positioned in vertex v_j at time step i, while ε^i_j is TRUE if and only if v_j is vacant at time step i.

The following constraints may then be used to model the validity conditions on two consecutive arrangements:

(1) $\bigwedge^{\mu}_{k,h=1,k<h} \neg\mathcal{X}^i_{j,k} \vee \neg\mathcal{X}^i_{j,h}$
for every $i \in \{1, \ldots, \eta\}$ and $j \in \{1, \ldots, n\}$

(at most one agent is placed in each vertex at each time step)

(2) $\mathcal{X}^i_{j,k} \Rightarrow \mathcal{X}^{i+1}_{j,k} \vee \bigvee_{l:\{v_j,v_l\}\in E} \mathcal{X}^{i+1}_{l,k}$
for every $i \in \{1, \ldots, \eta - 1\}, j \in \{1, \ldots, n\}$ and $k \in \{1, \ldots, \mu\}$

(an agent moves along an edge, or not at all)

(3) $\mathcal{X}^i_{j,k} \wedge \mathcal{X}^{i+1}_{l,k} \Rightarrow \varepsilon^i_l \wedge \varepsilon^{i+1}_j$

for every $i \in \{1, \ldots, \eta - 1\}, j, l \in \{1, \ldots, n\}$ such that $\{v_j, v_l\} \in E$ and $k \in \{1, \ldots, \mu\}$

(the target vertex is vacant before the move, and the source vertex will be vacant after it)

(4) $\varepsilon^i_j \Rightarrow \bigwedge_{h=1}^{\mu} \neg \mathcal{X}^i_{j,h}$

for every $i \in \{1, \ldots, \eta\}$, $j \in \{1, \ldots, n\}$

(relation between variables)

The instance's initial arrangement, α_0, is expressed through the following constraints:

(5) $\mathcal{X}^0_{j,k}$ for $v_j \in V$ if there is $a_k \in A$ such that $\alpha_0(a_k) = v_j$

$\neg \mathcal{X}^0_{j,k}$ otherwise

Likewise, the agents goal positions in the environment, α^+, are encoded in the constraints:

(6) $\mathcal{X}^\eta_{j,k}$ for $v_j \in V$ if there is $a_k \in A$ such that $\alpha^+(a_k) = v_j$

$\neg \mathcal{X}^\eta_{j,k}$ otherwise

3.2 Sequential Increasing Search

In order to get an optimal solution for an instance of CPF, we repeatedly try to find a solution of makespan η, for varying values of η. The simplest approach, sequential increasing search, or UNSAT-SAT, is described in Algorithm 1. It consists in repeatedly testing the instance for sequentially increasing values of η, starting with a Lower Bound (LB), for example, $\eta = 0$ (initial distribution identical to the final distribution). The first η for which a solution can be found is the cost of an optimal solution. For this process to be complete, an Upper Bound (UB) for the makespan must also be defined, otherwise the computation would not terminate when dealing with an unsolvable instance.

Algorithm 1 Find Optimal Solution using Sequential Increasing Search

Input: A CPF instance Σ
Output: An optimal solution for the instance

```
η ← LB
while η ≤ UB do
    F(Σ, η) ← encode_as_SAT(Σ, η)
    if solve_SAT(F(Σ, η)) then
        S ← extract_solution(F(Σ, η))
        return S
    η ← η + 1
return ∅
```

3.3 Binary Search

Alternatively, as a way to reduce the number of necessary iterations, a binary search can be used [4]. The instance is first tested for makespan $\eta = avg(UB, LB)$. Then, depending on whether or not a solution is found, the bounds of the search are updated. If a solution is found for this makespan, then the lower bound is updated: $LB = \eta + 1$; otherwise, the upper bound is updated: $UB = \eta$. When the upper and lower bounds have the same value, the last value of η for which a solution was found is the cost of the optimal solution. The pseudo-code is shown as Algorithm 2

Algorithm 2 Find Optimal Solution using Binary Search

Input: A CPF instance Σ
Output: An optimal solution for the instance

```
η ← avg(LB, UB)
S ← ∅
while LB ≠ UB do
    F(Σ, η) ← encode_as_SAT(Σ, η)
    if solve_SAT(F(Σ, η)) then
        S ← extract_solution(F(Σ, η))
        UB ← η
    else
        LB ← η + 1
    η ← avg(LB, UB)
return S
```

Sequential increasing search will, in the worst case scenario – either the instance is unsolvable or its optimal solution has makespan $\eta > UB$ – require UB iterations to finish, while binary search requires no more than $log_2(UB)$. However, given that the problem's complexity grows significantly with the makespan for which the instance is being tested, it is worth noticing that the sequential increasing search will never try to find a solution for a makespan higher than the optimal. Additionally, sequential increasing search will completely exploit the SAT-solver's incremental solving capabilities.

Both searches can be improved by a simple process, done only once for each instance – find the greatest distance between an agent and its goal, i. e., the length of the shortest possible path for the agent farthest away from its goal to reach it. This can be accomplished simply by running a series of μ breadth first searches, to find each agent's distance to its goal. By doing this, we can also immediately render unsolvable any instance on which one agent does not have a possible path to its goal (when the environment graph is not connected).

3.4 Independence Detection

CPF has exponential time complexity in the number of agents. To try to overcome this growth in difficulty, a method called Independence Detection (ID) was proposed by T. S. Standley in 2010 [10]. This method tries to identify the smallest possible group of agents for which paths can be found independently of all other agents of the instance. This way, the original problem is divided into a series of sub problems, each of which can be solved separately requiring a much lower computational effort. In the end, all the sub problems' solutions are combined, originating an optimal solution for the original problem. Even though this method was originally proposed to be used with a search-based solver, it was later adapted by P. Surynek in 2017 to be used alongside a SAT-solver [17].

The ID method is as following: Initially each agent is assigned to a group so that there are μ groups consisting of exactly one agent. Then, for each of these groups, an optimal solution is found independently. Every pair of these solutions is evaluated and if the two groups' solutions are in conflict (that is, when collision of agents belonging to different group occurs), the groups are merged and re-planned together. If there are no conflicting solutions, the solutions can be merged to a single solution of the original problem.

This method can be improved by considering that each group often allows for more than one optimal set of paths for its agents. When two groups' solutions are in conflict, instead of merging the groups right away, we try to replan one of them, adding clauses so that the states that originally caused the collision, as well as any others that might cause a new collision, are now forbidden. This way, not only the current collision is solved but we also avoid collisions with any of the already planned groups. If such replanning proves unsuccessful, the same method is tried for the other conflicting group. Only having both attempts failed are the groups merged. Pseudo-code for the complete ID method is shown as Algorithm 3.

Algorithm 3 Independence detection method

Input: A CPF instance Σ
Output: An optimal solution for the instance

```
assign each agent to a group
plan a path for each group G_1, ..., G_k
for each conflicting pair of groups do
    G_1, G_2 ←conflicting groups
    replan G_1 with illegal moves based on all other groups
    if failed to replan G_1 then
        replan G_2 with illegal moves based on all other groups
    if there are no alternate paths for G_1, G_2 then
        merge G_1 and G_2
        plan a path for the new group
return combined paths of all groups
```

4 Experiments

All the experiments were run in a Linux environment, using a 1.4GHz Processor, a maximum of 8GB of RAM and a timeout of 10 minutes. Glucose 4.1 Simple Solver [1] was used to solve all SAT queries.

First, a simple implementation of the solver was tested on 1350 4-connected grids randomly generated using Pavel Surynek's grid generator [15]. These grids may have obstacles, which correspond to the absence of a vertex in the environment graph. The agent occupancy rate on these grids ranges from 25% to 75%, and the obstacle rate is 10%. A third of these grids are of size 4x4 (4 to 12 agents), another third are 8x8 (16 to 48 agents), and the last are 16x16 (64 to 192 agents). SIMPLIFIED encoding and Sequential Increasing Search were used. The results are shown on Figure 4. 16x16 grids are not represented because none of them could be solved due to memory overflow.

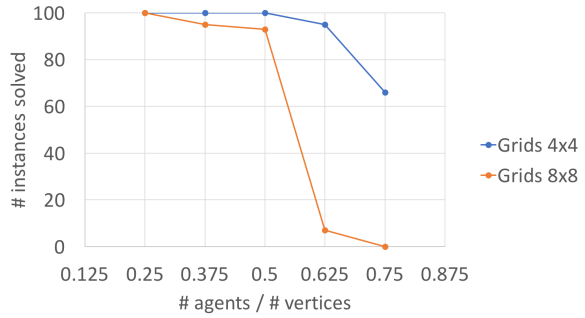
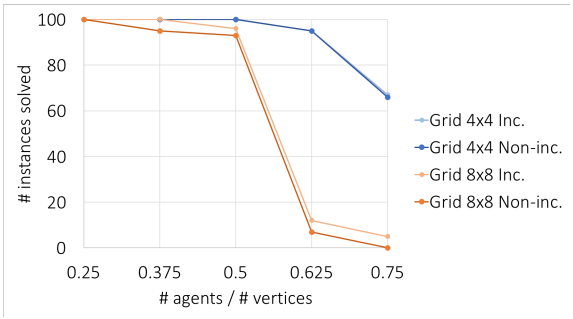
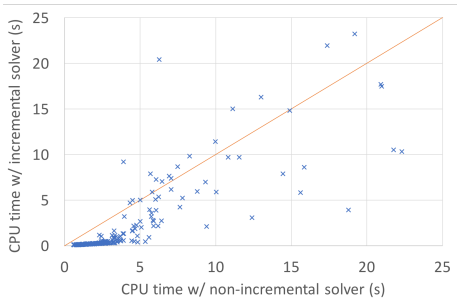


Figure 4: Number of instances solved on 4x4 and 8x8 4-connected grids by agent occupancy.

After adapting the solver to integrate Glucose's incremental solving capabilities, maintaining SIMPLIFIED encoding and Sequential Increasing Search, there were significant improvements to each instance's CPU solving time. The results can be seen in Figure 5. This approach has the advantage of keeping relevant information between calls to the SAT solver, potentially reducing the search space on the following iterations.



(a) Number of instances solved on 4x4 and 8x8 4-connected grids by agent occupancy, before and after applying the SAT solver's incremental capability.

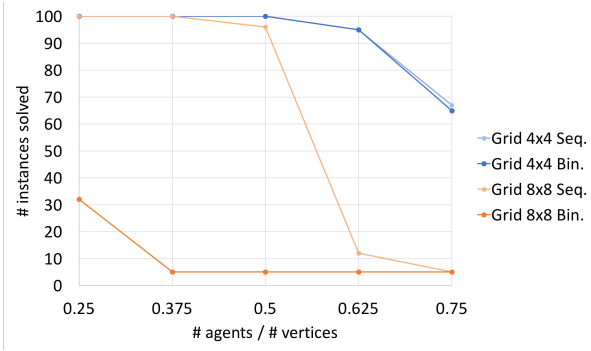


(b) Comparison of solving CPU times before and after applying the SAT solver's incremental capability. $y = x$ is represented in orange to help interpretation.

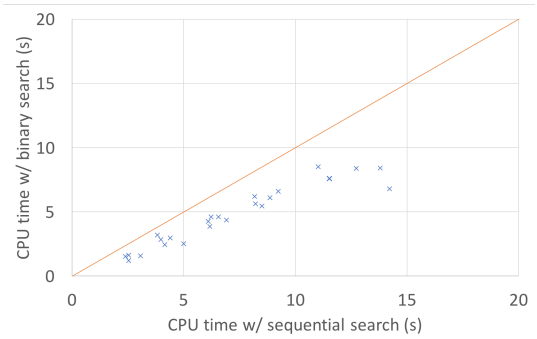
Figure 5: Incremental solving performance analysis.

Different search methods were tested as well. Binary search had a significantly worse performance than sequential increasing search, as can be seen on Figure 6. Analysing

the execution allowed us to conclude that this was due to the fact that the binary search started with a very high makespan value, which caused the first iteration of the search to be very slow – the search space grows exponentially with the makespan.



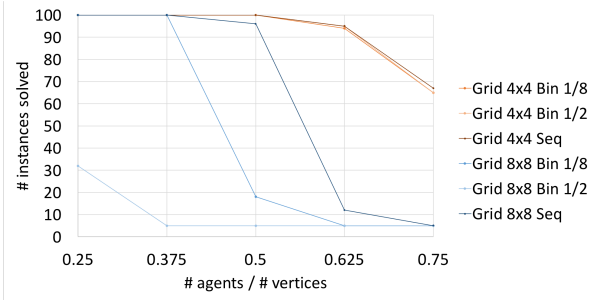
(a) Number of instances solved on 4x4 and 8x8 4-connected grids by agent occupancy, using binary search and sequential increasing search



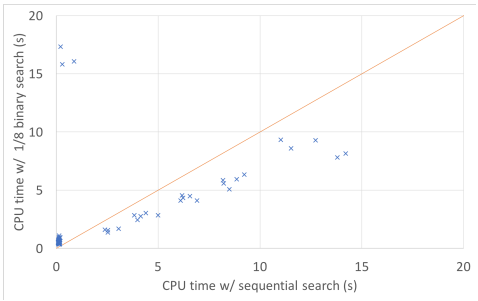
(b) Comparison of solving CPU times using binary and sequential increasing searches. $y = x$ is represented in orange to help interpretation.

Figure 6: Binary and sequential increasing searches performance analysis.

To try to compensate for this, a variation of the binary search was tested – instead of starting the search halfway between the upper and lower bounds, we tried starting it $1/8^{\text{th}}$ of the way. This produced better results, but still not as good as those achieved with sequential increasing search, as can be seen on Figure 7.



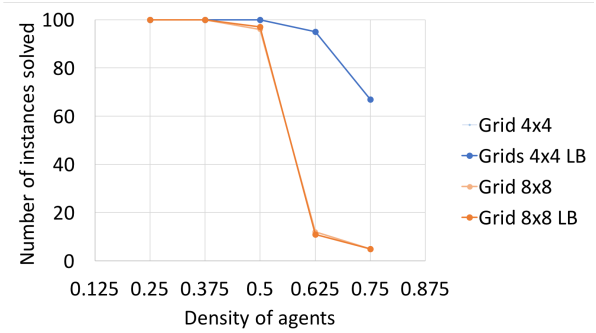
(a) Number of solved 4x4 and 8x8 4-connected grids by agent occupancy, using binary search, binary $1/8^{\text{th}}$ variation and sequential increasing search.



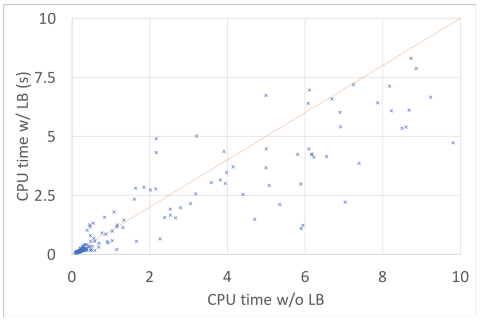
(b) Comparison of solving CPU times using binary $1/8^{\text{th}}$ variation and sequential increasing searches. $y = x$ is represented in orange to help interpretation.

Figure 7: Binary, binary $1/8^{\text{th}}$ variation and sequential increasing searches performance analysis.

Afterwards, the computation of a lower bound was tested. Before starting the search, we measured each agent's distance to its goal using a series of breadth first searches, and used the highest distance as a lower bound for the search. Furthermore, if there were any agent with no possible path to its goal (the graph is disconnected), the instance can be instantly rendered unsolvable. The number of solved instances did not increase but the solving time showed some improvement, as can be seen on Figure 8.



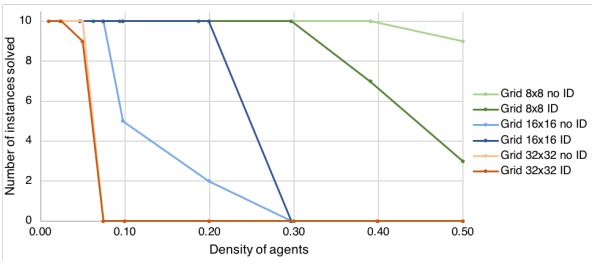
(a) Number of solved 4x4 and 8x8 4-connected grids by agent occupancy, before and after computing a higher value for the search's lower bound.



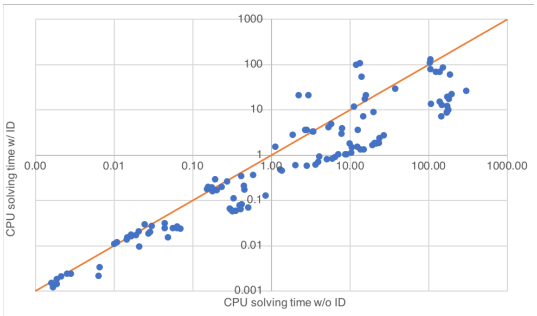
(b) Comparison of solving CPU times before and after computing a higher value for the search's lower bound. $y = x$ is represented in orange to help interpretation.

Figure 8: Lower bound computation performance analysis.

To test the independence detection method a different set of 260 instances were used, with finer granularity between occupancy rates, and up to 40%. Using this method, we were for the first time able to solve some larger grids with low occupancy rates: 16x16 and 32x32 grids with occupancies up to 20% and 5% respectively. Using ID caused slight improvement in the performance in instances up to 20% occupancy in the 4x4 and 8x8 grids, and much worse performance in higher occupancy rates. These results are shown on Figure 9.



(a) Number of solved 8x8, 16x16 and 32x32 4-connected grids by agent occupancy, with and without employing the ID method. 4x4 instances are not represented because all of them were solved with and without ID.



(b) CPU solving time comparison with and without ID tested on 4-connected grids of size 4x4 up to 32x32, with occupancy rates up to 20%. $y = x$ is represented in orange to help interpretation.

Figure 9: ID method performance analysis.

5 Conclusions

In this project, we studied and evaluated several different SAT-based techniques to find an optimal solution for the problem of cooperative path-finding.

A CPF solver was developed using the SIMPLIFIED encoding to translate the CPF problem to SAT, which was then modified: incremental solving was incorporated, sequential increasing search was compared with binary search, and tested alongside other methodologies, such as computing a higher lower bound before starting the search. Finally, the independence detection method was incorporated and evaluated.

Incremental solving proved to be a valuable tool, as it significantly reduced the search time in most cases. Sequential increasing search consistently outperformed binary search, even after modifying the latter to start the search closer to the optimal makespan value. Computing the agents' distances to their goals also proved fruitful and it reduced the total solving time . Applying ID is advantageous in instances with less than 20% occupancy, causing an increase in solving time in more crowded environments.

Overall, solving methods based on propositional satisfiability emerge as valuable tools to efficiently find optimal solutions to cooperative path-finding.

6 Acknowledgements

I would like to thank the Calouste Gulbenkian Foundation, who supported this project, for offering me a place on the first edition of the New Talents in Artificial Intelligence Grant and the invaluable opportunity to be a part of their outstanding network of students and academics.

I would also like to express my gratitude to Professor Inês Lynce, for sharing expertise, valuable guidance, and for the continuous encouragement she extended to me.

References

[1] Gilles Audemard and Laurent Simon. The Glucose SAT Solver. `http://www.labri.fr/perso/lsimon/glucose`, 2014. Accessed: September 2018.

[2] Kurt M. Dresner and Peter Stone. A multiagent approach to autonomous intersection management. *J. Artif. Intell. Res.*, 31:591–656, 2008.

[3] Ariel Felner, Roni Stern, Solomon Eyal Shimony, Eli Boyarski, Meir Goldenberg, Guni Sharon, Nathan R. Sturtevant, Glenn Wagner, and Pavel Surynek. Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In Alex Fukunaga and Akihiro Kishimoto, editors, *Proceedings of the Tenth International Symposium on Combinatorial Search, Edited by Alex Fukunaga and Akihiro Kishimoto, 16-17 June 2017, Pittsburgh, Pennsylvania, USA.*, pages 29–37. AAAI Press, 2017.

[4] António Morgado, Federico Heras, Mark H. Liffiton, Jordi Planes, and João Marques-Silva. Iterative and core-guided maxsat solving: A survey and assessment. *Constraints*, 18(4):478–534, 2013.

[5] Robert Morris, Corina S. Pasareanu, Kasper Søe Luckow, Waqar Malik, Hang Ma, T. K. Satish Kumar, and Sven Koenig. Planning, scheduling and monitoring for airport surface operations. In Daniele Magazzeni, Scott Sanner, and Sylvie Thiébaux, editors, *Planning for Hybrid Systems, Papers from the 2016 AAAI Workshop, Phoenix, Arizona, USA, February 13, 2016.*, volume WS-16-12 of *AAAI Workshops*. AAAI Press, 2016.

[6] Van Nguyen, Philipp Obermeier, Tran Cao Son, Torsten Schaub, and William Yeoh. Generalized target assignment and path finding using answer set programming. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 1216–1223. ijcai.org, 2017.

[7] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artif. Intell.*, 219:40–66, 2015.

[8] Guni Sharon, Roni Stern, Meir Goldenberg, and Ariel Felner. The increasing cost tree search for optimal multi-agent pathfinding. *Artif. Intell.*, 195:470–495, 2013.

[9] David Silver. Cooperative pathfinding. In R. Michael Young and John E. Laird, editors, *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference, June 1-5, 2005, Marina del Rey, California, USA*, pages 117–122. AAAI Press, 2005.

[10] Trevor Scott Standley. Finding optimal solutions to cooperative pathfinding problems. In Maria Fox and David Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, 2010.

[11] Pavel Surynek. An optimization variant of multi-robot path planning is intractable. In Maria Fox and David Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, 2010.

[12] Pavel Surynek. A simple approach to solving cooperative path-finding as propositional satisfiability works well. In Duc Nghia Pham and Seong-Bae Park, editors, *PRICAI 2014: Trends in Artificial Intelligence - 13th Pacific Rim International Conference on Artificial Intelligence, Gold Coast, QLD, Australia, December 1-5, 2014. Proceedings*, volume 8862 of *Lecture Notes in Computer Science*, pages 827–833. Springer, 2014.

[13] Pavel Surynek. Simple direct propositional encoding of cooperative path finding simplified yet more. In Alexander F. Gelbukh, Félix Castro-Espinoza, and Sofía N. Galicia-Haro, editors, *Nature-Inspired Computation and Machine Learning - 13th Mexican International Conference on Artificial Intelligence, MICAI 2014, Tuxtla Gutiérrez, Mexico, November 16-22, 2014. Proceedings, Part II*, volume 8857 of *Lecture Notes in Computer Science*, pages 410–425. Springer, 2014.

[14] Pavel Surynek. Makespan optimal solving of cooperative path-finding via reductions to propositional satisfiability. *CoRR*, abs/1610.05452, 2016.

[15] Pavel Surynek. Annex to integration of independence detection into sat-based optimal multi-agent path finding: A novel sat-based optimal mapf solver. `http://ktiml.mff.cuni.cz/~surynek/research/icaart2017`, 2017. Accessed: September 2018.

[16] Pavel Surynek and Petr Michalík. Improvements in sub-optimal solving of the (n^2-1)-puzzle via joint relocation of pebbles and its applications to rule-based cooperative path-finding. *CoRR*, abs/1610.04964, 2016.

[17] Pavel Surynek, Jiri Svancara, Ariel Felner, and Eli Boyarski. Integration of independence detection into sat-based optimal multi-agent path finding - A novel sat-based optimal MAPF solver. In H. Jaap van den Herik, Ana Paula Rocha, and Joaquim Filipe, editors, *Proceedings of the 9th International Conference on Agents and Artificial Intelligence, ICAART 2017, Volume 2, Porto, Portugal, February 24-26, 2017.*, pages 85–95. SciTePress, 2017.

[18] Manuela M. Veloso, Joydeep Biswas, Brian Coltin, and Stephanie Rosenthal. Cobots: Robust symbiotic autonomous mobile service robots. In Qiang Yang and Michael Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, page 4423. AAAI Press, 2015.

[19] Peter R. Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, 29(1):9–20, 2008.

[20] Jingjin Yu and Steven M. LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In Marie desJardins and Michael L. Littman, editors, *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*. AAAI Press, 2013.