ENS 491-492 – Graduation Project

Final Report

Project Title: Gossip Problems

Group Members:

25309 Haşim Sait Göktan

25421 Berkin Karaçam

23870 Zeki Oğuzhan İnce

24200 Taygun Özcan

Supervisor(s): Esra Erdem

Date: 30.05.2021

Sabancı Üniversitesi

- **ENS 491-492 Final Report must contain the following sections (leave section titles unchanged).**
- **You are required to write clearly and concisely: The main body of the report (not including the Title Page, Appendix, and References) must be between 20-25 pages.**
- **Project advisor may have additional requirements for the report.**

1. EXECUTIVE SUMMARY

The gossip problem is an information distribution problem which involves some amount of people. All of the agents know a piece of information which is not known by other agents. They communicate by telephone and when two agents talk with each other, they share all gossips that they know at that time. In the beginning, all agents only know their own secrets. In the goal state, each person must know all the gossips. So, the problem aims to find the minimum number of calls needed to provide all of the information to all of the agents.

There are different variations of the gossip problem. The most common variation is an optimization problem which aims to minimize the number of calls needed to provide all of the information to all of the people. Some of the other variations include negative goals, in which some agents should not know the secrets of some other agents, and directional gossips, where communication does not have to be bi-directional, and is limited from some agents to some other agents. The variations of the gossip problems with negative goals and directional gossips are intractable.

The motivation behind this project is to develop and evaluate computational methods to solve gossip problems with a valid background on logic and algorithms. The gossip problem has been popularly focused on by different mathematicians. One of the papers that we focus on is "Gossips and Telephones" by Brenda Baker and Robert Shostak. In this paper, they focus on the main version of the gossip problems in which the aim is to minimize the needed number of telephone calls that provide all of the agents with all of the information. Our difference is, with this project we focus not only on the main version, but also on different variations of gossip problems, using AI tools.

AI methods such as search, constraint programming and other declarative methods are used with a solid plan to solve each variation of this problem. To achieve this goal, these methods are compared empirically. The problem that the project focuses on is a complex problem since it does not have a simple/obvious solution. It requires the use of different models of AI and specialties of computer science/engineering because it involves various components and conflicting multiple realistic constraints. So, we must work experimentally with an analytical approach by making research to produce solutions with help of our fundamental knowledge.

2. PROBLEM STATEMENT

The gossip problem is an information distribution problem which involves some amount of people. All of the agents know a piece of information which is not known by other agents. They communicate by telephone and when two agents talk with each other, they share all gossips that they know at that time. In the beginning, all agents only know their own secrets. In the goal state, each person must know all gossips. So, the problem aims to find the minimum number of calls needed to provide all of the information to all of the agents.

There are different variations of the gossip problem. The most common variation is an optimization problem which aims to minimize the number of calls needed to provide all of the information to all of the people. Some of the other variations include negative goals, in which some agents should not know the secrets of some other agents, and directional gossips, where communication does not have to be bi-directional, is limited from some agents to some other agents. The variations of the gossip problems with negative goals and directional gossips are intractable.

The motivation behind this project was to develop and evaluate computational methods to solve gossip problems with a valid background on logic and algorithm. The gossip problem has been popularly focused on by different mathematicians. One of the papers that we focused on was "Gossips and Telephones" by Brenda Baker and Robert Shostak. In this paper, they focus on the main version of the gossip problems in which the aim is to minimize the needed number of telephone calls that provide all of the agents with all of the information. Our difference is, with this project we focused not only on the main version, but also on different variations of gossip problems, using AI tools. We tried different search algorithms such as Uniform Cost Search, A* search and we tried to use our knowledge about Constraint Satisfaction Problems. Planning is another AI method that we tried to use in this project.

2.1. Objectives/Tasks

The objective of the project was to solve the gossip problem and its' variations using different AI methods and algorithms. For this, we were assigned with different tasks every week after our weekly meetings.

The tasks assigned and their results are given below:

1. Objective 1 / Task 1:
   Objective: Searching and understanding what gossip problems are.
   Task: Implementing the basic gossip problem in both Python and C++ languages.
   Result: The result of this task was the implementation of the problem in both languages.

2. Objective 2 / Task 2:
   Objective: Understanding what negative goals mean for gossip problems.
   Task: Implementing the variation of the gossip problem with negative goals.
   Result: The result of the task was the implementation of the gossip problem with negative goals in Python language.

3. Objective 3 / Task 3:
   Objective: Learning about constraint satisfaction problems and how we can turn our problem into this type of problem.

Task: Learning the Picat language using the videos and other sources provided in the website.

Result: The result of the task was completing watching the videos about Picat language and to gain more information about the language.

4. Objective 4 / Task 4:
   Objective: Thinking about how to solve different problems using Picat language, getting familiar with a new AI method, planning.
   Task: Looking at the different planning problems provided on the website such as 15-Puzzle and Sokoban
   Result: The result of the task was focusing more on the language and understanding more about planning.

5. Objective 5 / Task 5:
   Objective: Learning about how to use planning in implementation of the problem in FF.
   Task: Implementing both the basic gossip problem and its' variation with negative goals in FF.
   Result: The result of the task was the implementation of these problems in FF.

6. Objective 6 / Task 6:
   Objective: Learning about how to use planning in implementation of the problem in PDDL.
   Task: Implementing both the basic gossip problem and its' variation with negative goals in PDDL.
   Result: The result of the task was the implementation of these problems in PDDL with problem and domain descriptions.

7. Objective 7 / Task 7:
   Objective: Experimenting both the basic problem and the variation with negative goals using different planners.
   Task: Creating different problem descriptions and testing.
   Result: The result of the task was changing results in number of feasible solutions and average total time for each planner.
8. Objective 8 / Task 8:
   Objective: Learning about epistemic knowledge problem and how the gossip problem can be represented as an epistemic knowledge problem.
   Task: Reading two different articles about the gossip problem where the epistemic depth is bigger than 1.
   Result: The result of the task was completing the reading task and gaining more information about the gossip problem when epistemic depth increases.

9. Objective 9 / Task 9:
   Objective: Learning about the implementation of the problem when epistemic depth increases.
   Task: Implementing the gossip problem in PDDL when epistemic depth is equal to 2.
   Result: The result of the task was the new domain implementation for epistemic depth 2.

10. Objective 10 / Task 10:
    Objective: Experimenting the basic problem with depth 2 using different planners.
    Task: Testing different instances when epistemic depth is equal to 2.
    Result: The result of the task was changing results in average total time for each planner.

11. Objective 11 / Task 11:
    Objective: Learning about the implementation of the problem with negative goals when epistemic depth is equal to 2.
    Task: Thinking about how the negative goals should be included in the domain and problem descriptions and changing the domain and problem generators accordingly to create different instances and a new domain for the problem with negative goals when the depth is equal to 2.
    Result: The result of the task was the new domain and problem definitions for epistemic depth 2.

12. Objective 12 / Task 12:
    Objective: Experimenting the gossip problem with negative goals using different planners when depth is equal to 2.
    Task: Testing different instances with negative goals when epistemic depth is equal to 2.
    Result: The result of the task was changing results in number of feasible solutions and average total time for each planner.

2.2. Realistic Constraints

Since the project was an implementation-based project, there were not any constraints regarding environment, economy and health and safety. However, in this project there were some realistic constraints such as maintainability and manufacturability.

The maintainability of this project is determined by the properties of its source code. Maintainability refers to the ease with which the project can be modified. Since we were working as a group of four people, we had to implement different code pieces and putting them together and modifying them properly caused some difficulties for us. Even though we did not create a physical product, manufacturability was one of the constraints that needed our focus. Manufacturability refers to the ability to produce an effective product satisfying its quality, cost and time requirements. Since we did not produce a physical product, we mostly had time and quality requirements and therefore we had to deal with manufacturability constraint regarding these constraints.

- Maintainability
  In the project, we were working as a group of four people, and we sometimes had to implement different code pieces. While we were doing these, codes needed to be readable, changeable and understandable because we had to understand and follow each other's code pieces to maintain the project. Also, since none of us were experienced in AI methods before, especially planning, we had to implement in different programming languages that took a lot of time to get used to.

- Manufacturability
  Working time of the written code was needed to be appropriate in terms of working time. In other words, after the implementation, complexity time of the code needed to be reachable in order to test. For example, in the implementation of the problem with negative goals, the given size of the people and the percentage of negative goals played a big role while we were testing the code, when we tested on 5 people it worked properly but when we increased the number of people to 6 or 7, we usually did not have enough time and RAM to test the code because the algorithms worked in exponential time.

3. METHODOLOGY

The project required us to use different methods of AI before reaching the results. The main two methods were search and planning. We had many different tasks regarding the implementation of these methods. We used different languages and performed many experiments in order to reach the results.

Implementation of the Basic Gossip Problem:

This project aimed to produce solutions for different variations of gossip problems and implement algorithms with reasonable time complexity. For instance, as the first task of the project, we implemented an algorithm to solve the basic gossip problem (no constraints, everybody should know all secrets). There are n people, we set any four people as chiefs and one of the chiefs as a leader. At first, the leader talks with everybody except chiefs. In the second step, chiefs talk with each other. In the third and the last step, the leader talks with everybody except chief again. So, in final state everybody knows all secrets with 2n-4 calls(n>4). The pseudocode for the first task:

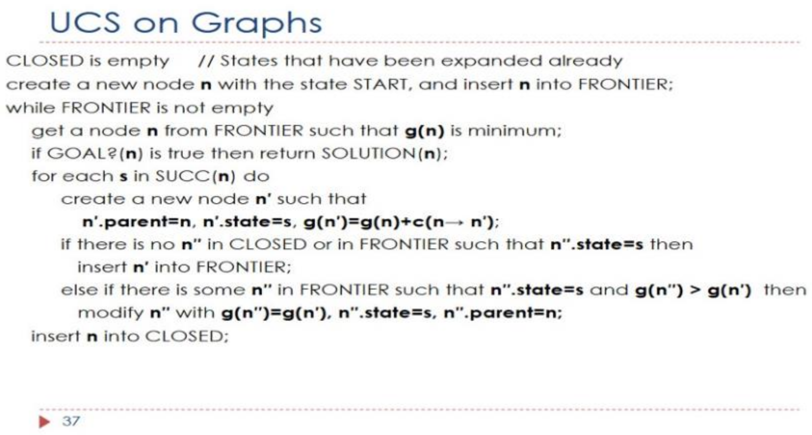
```
gossipProblem {
          int people[n]; # n agents, the first 4 are the chiefs, 4th one is the leader
          int count = 0; # will calculate the total number of calls
          # First step, 4th chief calls people except chiefs.
          for (i = 4: n) { # i will be equal to all numbers from 5 to n, the leader will call all
                    exchangeSecrets (3, i); # the call between the leader and the i'th agent
                    count += 1;
```

```
        }
        # Second step, 4 chiefs call each other
        exchangeSecrets (0, 1);
        exchangeSecrets (2, 3);
        exchangeSecrets (0, 2);
        exchangeSecrets (1, 3);
        count += 4;
        # Third(last) step, the leader calls everyone except the other chiefs
        for (i = 4: n) {
                exchangeSecrets (3, i);
                count += 1;
        }
}
```

This algorithm solves the gossip problem in polynomial time.

Implementation of the Gossip Problem with Negative Goals:

As the second task of the project, we were asked to implement the solution to one of the variations of the problem. This variation included negative goals, where some agents should not know the secrets of some other agents. We completed the implementation using Python language. At first, we turned this problem into a search problem using what we learned during our CS 404 lectures. We tried three different search algorithms on the problem, depth first search, A* search and uniform cost search. At the end, we decided to use uniform cost search for the solution to the problem. The pseudocode of uniform cost search algorithm is given below:



UCS on Graphs

```
CLOSED is empty     // States that have been expanded already
create a new node n with the state START, and insert n into FRONTIER;
while FRONTIER is not empty
    get a node n from FRONTIER such that g(n) is minimum;
    if GOAL?(n) is true then return SOLUTION(n);
    for each s in SUCC(n) do
        create a new node n' such that
          n'.parent=n, n'.state=s, g(n')=g(n)+c(n→ n');
        if there is no n" in CLOSED or in FRONTIER such that n".state=s then
          insert n' into FRONTIER;
        else if there is some n" in FRONTIER such that n".state=s and g(n") > g(n') then
          modify n" with g(n")=g(n'), n".state=s, n".parent=n;
    insert n into CLOSED;
```

▶ 37

Planning in PDDL

After search, the main focus of the project was planning. Planning is another AI method, which was crucial for this project. In planning problems, it is aimed to reach a goal state from an initial state. We were asked to formulate the gossip problem as a planning problem using PDDL, Planning Domain Definition Language.

In PDDL, a problem description and a domain description were created. In the domain description, a fluent and an action were defined. In AI, a fluent is a condition that can change over time. For our domain description, defining one fluent was enough, which is defined as knows (a, s). Knows (a, s) gives the truth value of agent a knowing secret s or not. The only action we had was calling action. This action takes place between two different agents (a1, a2) and with this action, it is aimed to share all of the secrets that are currently known by a1 with a2 and to share the secrets of a2 with a1. The action was defined using ADL, Action Description Language. The most important features of ADL are enabling us with typing and with conditional effects. With typing, we were able to distinguish the objects as agent objects and secret objects. Conditional effects are needed when we need to use the "when" keyword. In the action of calling, we look for all secrets. For each secret, when this secret is known by a1 or a2, after the action this secret will be known by both agents. The effect of the action has to be conditional because the action looks for all of the secrets and the number of agents who know a specific secret, changes after each step. Therefore, ADL was important for the action description in PDDL since typing and conditional effects were needed.

In the problem description, the objects; all of the agents and their secrets, and two states; the initial state and the goal state were defined. The initial state is where all of the agents know only their own secrets and the goal state is where all of the secrets are known by each agent. The initial state is defined using the fluent, knows (a, s). For every agent a, and their own secret s, the initial state includes the fluent knows (a, s) because at the beginning, the agents only know their own secrets. Therefore, in the case of having 6 agents and their 6 secrets, the initial state includes the fluent knows (a, s) 6 times: knows (a1, s1), knows (a2, s2), …, knows (a6, s6). The goal state on the other hand includes the fluent 36 times because the goal of the plan is to reach a state where all of the secrets are known by each agent.

In order to reach the goal state from the initial state, a plan is needed in which each step includes the action of calling. The planner that we provided with the PDDL definitions, gave a plan as the output that enabled us to reach the goal state from the initial state. This plan included a number of steps of calling.

Testing with Different Planners

Each group member was asked to use different planners to test and record the results for the basic problem, the problem with negative goals, the basic problem with epistemic depth 2 and lastly, the problem with negative goals when the depth is equal to 2.

Haşim Sait Göktan used Planning-PDBs, Berkin Karaçam used METIS1, Zeki Oğuzhan İnce used COMPLEMENTARY1 and Taygun Özcan used SCORPION planners. The results of the tests are given under Results & Discussion.

4. RESULTS & DISCUSSION

All of the initial objectives were realized, and the project is successfully completed. Different results for different tasks that were assigned during the project are given below:

Test Results with Epistemic Depth 1, Both the Basic Problem and With Negative Goals

1. Planning-PDBs --- Haşim Sait Göktan

All planners are built using Singularity containers by following the instructions given on https://ipc2018-classical.bitbucket.io.

A domain file with "forall" enabled us to use a single domain definition for all problem instances.

Memory and time limits are added using the commands "ulimit -t 1800" and "ulimit -v 8388608" as provided in https://ipc2018-classical.bitbucket.io.

Random problem instances with uniformly distributed negative goals are generated, then the generated instances are tested for feasibility using Planning-PDBs. If the problem instance is infeasible, the command that solves the instance returns false and a new random instance is generated.

```
$while! singularity run-C-H $RUNDIR planner.img $DOMAIN" $(pwd)/problem_n_x.pddl" $RUNDIR/sas_plan; dopython3path/to/problem_generator.py n x; done
```

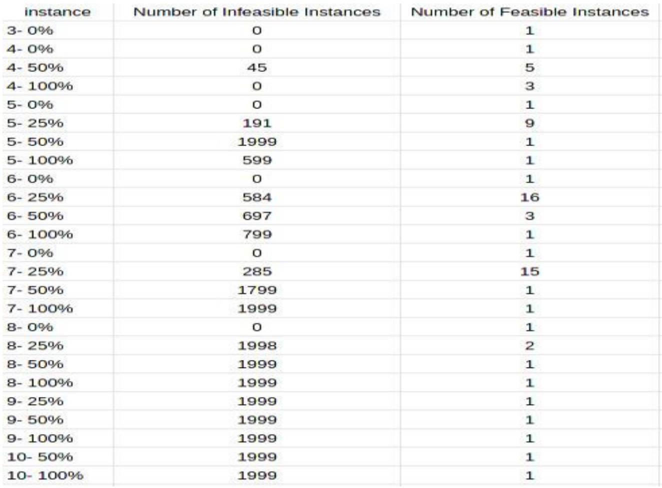
Once this command terminates, a feasible problem instance is saved at "problem_n_x.pddl".

For the n x pairs where the command does not terminate within 1 hour, a problem instance is created by manually editing a problem instance to be feasible with attention to keep the distribution as uniform as possible.

X value represents the (number of negative goals per agent) / (number of agents - 2). X is 1 when there are n-2 negative goals per agent, 0 when there are no negative goals.

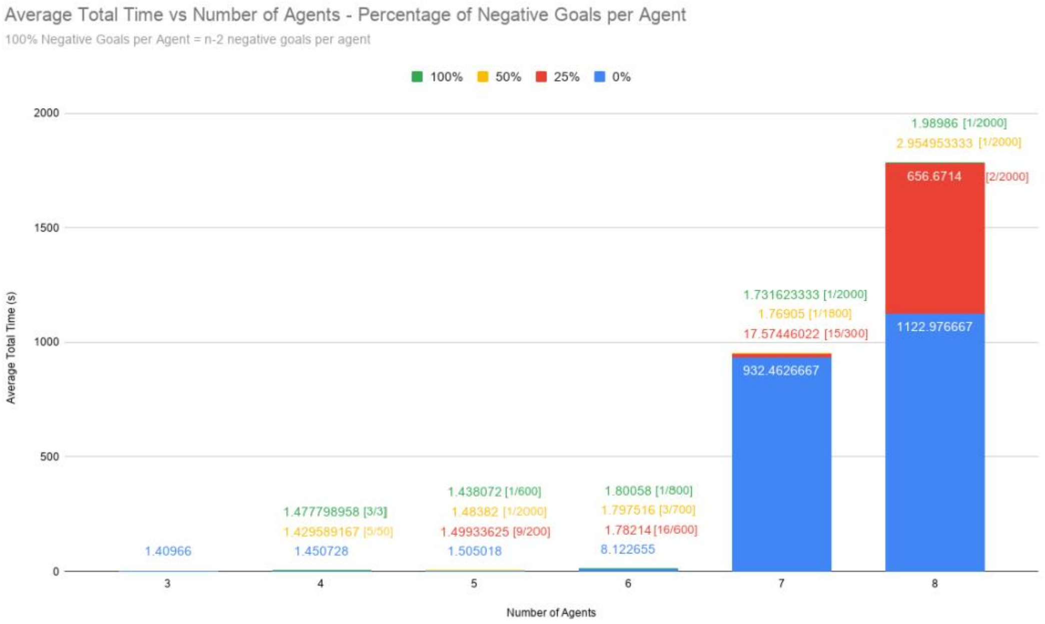
Then each feasible problem instance is solved by Planning-PDBs 10 to 3 times (depending on the total time).

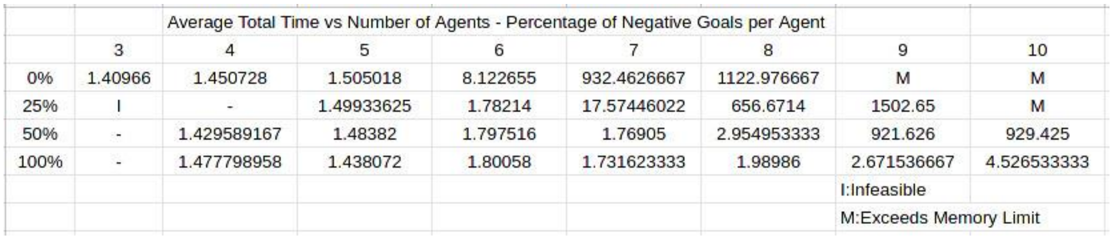
Solutions and total times are logged, the average of total times and length of solutions are presented in the graphs and tables below:



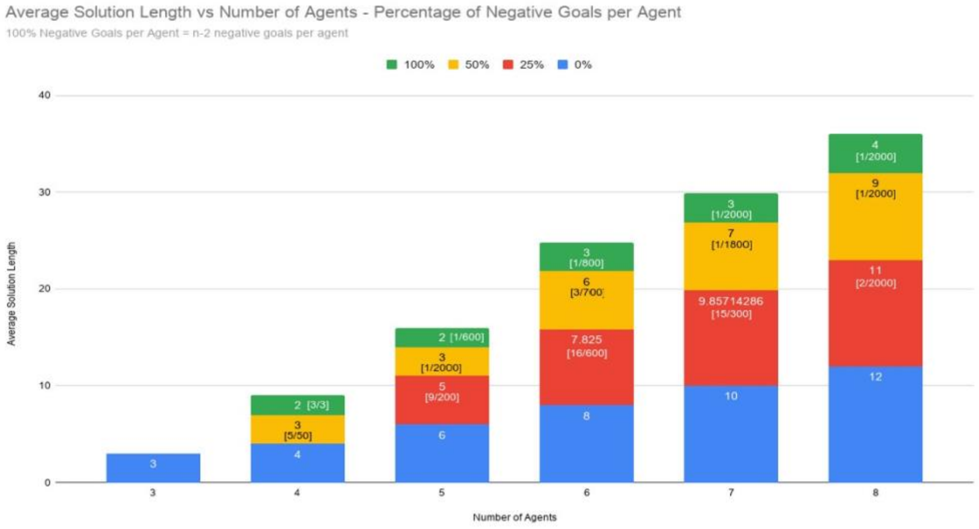
instance	Number of Infeasible Instances	Number of Feasible Instances
3- 0%	0	1
4- 0%	0	1
4- 50%	45	5
4- 100%	0	3
5- 0%	0	1
5- 25%	191	9
5- 50%	1999	1
5- 100%	599	1
6- 0%	0	1
6- 25%	584	16
6- 50%	697	3
6- 100%	799	1
7- 0%	0	1
7- 25%	285	15
7- 50%	1799	1
7- 100%	1999	1
8- 0%	0	1
8- 25%	1998	2
8- 50%	1999	1
8- 100%	1999	1
9- 25%	1999	1
9- 50%	1999	1
9- 100%	1999	1
10- 50%	1999	1
10- 100%	1999	1

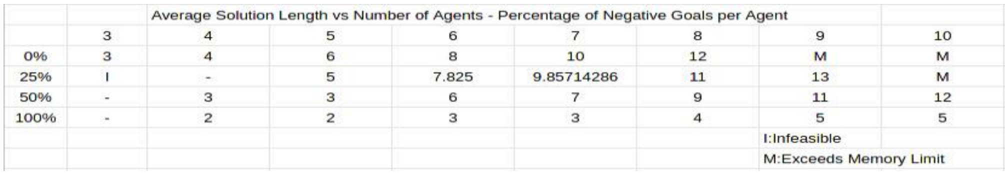
The values in the square brackets in the below graphs show:
[Number of feasible instances/Number of total instances]





		Average Total Time vs Number of Agents - Percentage of Negative Goals per Agent						
	3	4	5	6	7	8	9	10
0%	1.40966	1.450728	1.505018	8.122655	932.4626667	1122.976667	M	M
25%	I	-	1.49933625	1.78214	17.57446022	656.6714	1502.65	M
50%	-	1.429589167	1.48382	1.797516	1.76905	2.954953333	921.626	929.425
100%	-	1.477798958	1.438072	1.80058	1.731623333	1.98986	2.671536667	4.526533333
							I:Infeasible	
							M:Exceeds Memory Limit	





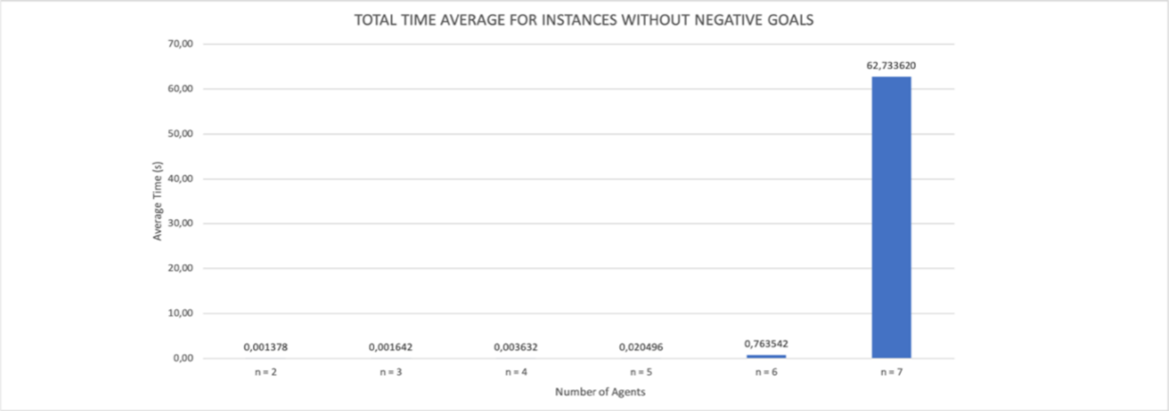
		Average Solution Length vs Number of Agents - Percentage of Negative Goals per Agent						
	3	4	5	6	7	8	9	10
0%	3	4	6	8	10	12	M	M
25%	I	-	5	7.825	9.85714286	11	13	M
50%	-	3	3	6	7	9	11	12
100%	-	2	2	3	3	4	5	5
							I:Infeasible	
							M:Exceeds Memory Limit	

2. METIS1 --- Berkin Karaçam

Each of the group members worked with different planners that were installed using Singularity containers, in order to test the total time needed for a planner to find a plan for different problem descriptions. The domain description did not change for the experiments, all of the members used the domain description that was written before using PDDL. Different problem descriptions were created using the problem generator that is in Haşim's directory. These descriptions may include different number of agents and negative goals with different percentages. The percentages of negative goals show how many negative goals an agent will have, for example, if the number of agents is 4 and the negative goal percentage is 25%, each of the agents will have 1 negative goal. For all of the test descriptions below, n is used for the number of agents and p is used for the percentages of negative goals, for simplicity.

First of all, since the aim is to create as many tests as possible, the tests are divided into two categories: the tests for the basic gossip problem without negative goals and the tests with negative goals. For the tests without negative goals, the aim is to provide each agent with all of the secrets. Therefore, these tests always provide a plan and will not result in any infeasible solutions.

The tests started for n = 2 and could continue until n = 7, METIS1 could not complete the execution for n = 8. Until n = 7, all of the time values were quite small and close to each other, METIS1 planner worked faster than expected. The graph for time vs n where p=0% is given below.



As it is observed in above graph, METIS1 worked very fast until n=7.

For tests with negative goals, randomly generated instances are used. Each case with n and p values is described below:

n=4, p=25%:

In this case each agent has one negative goal. 25 different instances are generated, and these instances are used to create a plan using the planner, however, only 4 of the instances resulted in feasible solutions. This step showed that it would be hard to generate feasible solutions for randomly generated instances.

n=4, p=50%:
Therefore, for n=4 and p=50%, where two negative goals exist for each agent, the instances that are generated in the first test are used as seeds. For each four of these seeds that resulted in feasible solutions, 10 different instances are generated. Out of these 40, only 2 were feasible.

n=5, p=25% and n=5, p=50%:
For n=5, the same steps that were applied for n=4 are used. First, the instances are generated randomly for p=25% and 2 feasible solutions are created out of 25. Then these 2 seeds are used to generate for p=50%. However, none of the 66 tries resulted in a feasible solution.

n=6, p=25%:
For n=6 and p=25%, first, random generation is used without using any seeds. After failing to find any feasible solutions in 25 attempts, the seeds that were generated for n=4 and p=25%, and n=5 and p=25% are used. This step resulted in 10 feasible solutions in 50 tries.

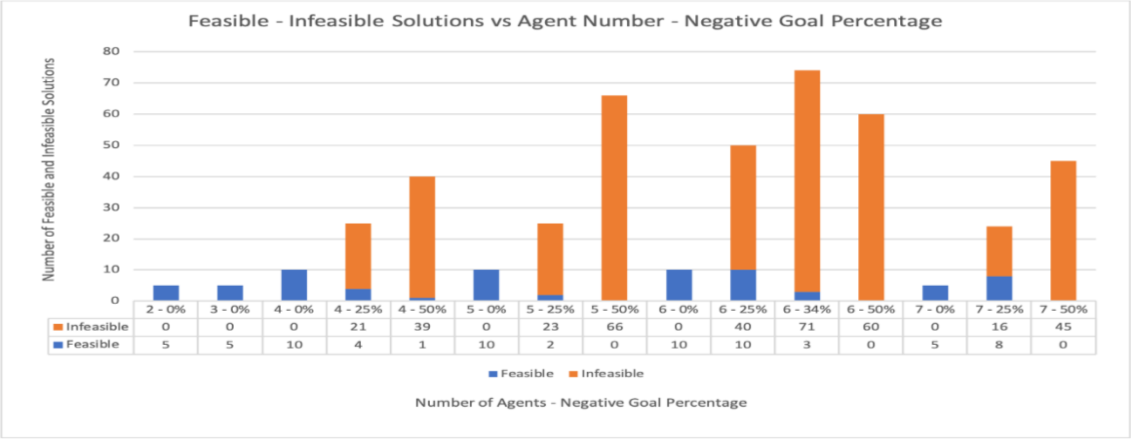
n=6, p=34%:
This test is created in order to have 2 negative goals as well for each agent. For this test if any seeds were not used the probability of finding a feasible solution would be very low. Therefore, the seeds that were produced for n=4 and p=50% are used. In 74 tries, it resulted in 3 different feasible solutions.

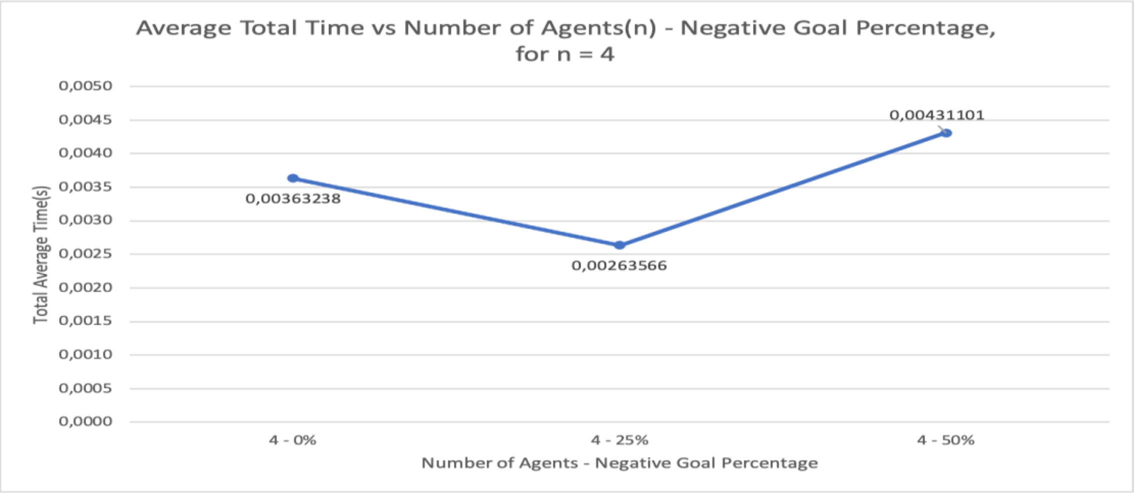
n=6, p=50%:
The tests continued with n=6 and p=50% by using the seeds that were produced for p=34%. However, none of the 60 tries resulted in a feasible solution.

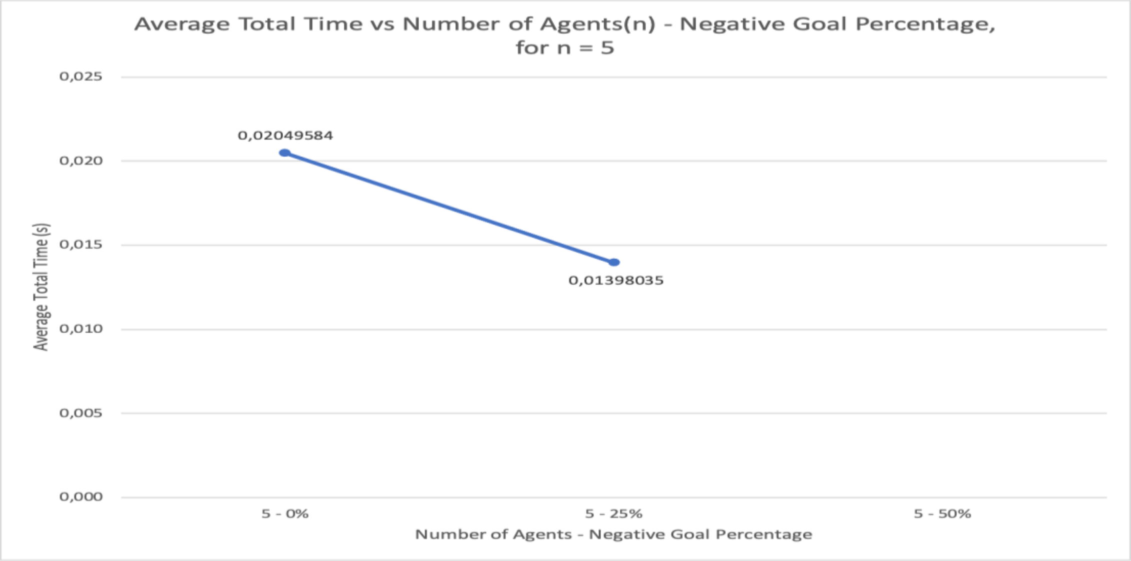
n=7, p=25%:
Then for n=7 and p=25%, four of the instances from n = 6 and p = 25% are used as seeds. For each four of these seeds, there are 6 different probabilities for negative goal generation because in this step one negative goal is needed for each agent. In this step, each 24 possibility is tried, and it resulted in 8 feasible solutions.

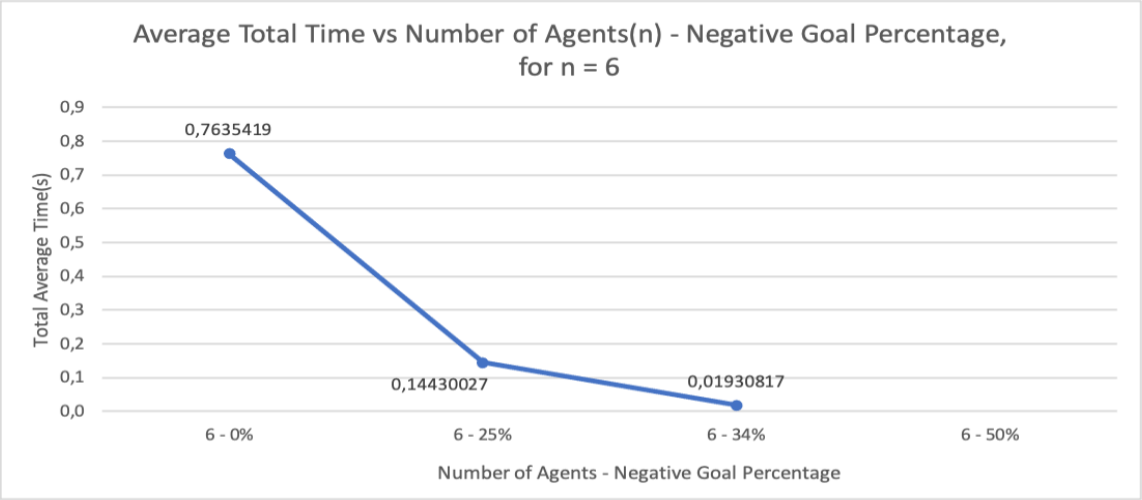
n=7, p=50%:
Lastly, 8 seeds that were generated for n=7, p=25% are used. However, this did not result in any feasible solutions.

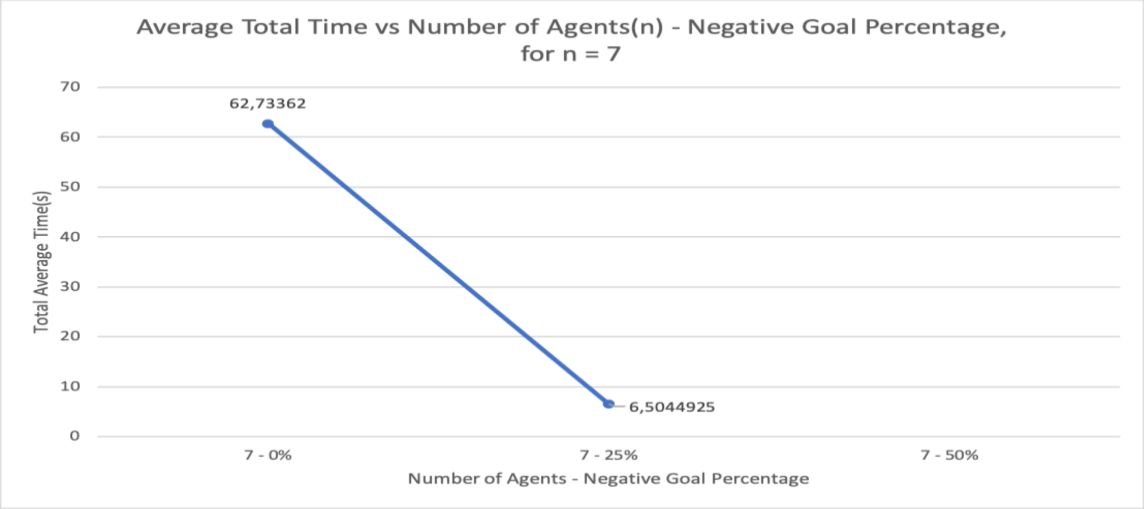
The graphs that visualize the number of feasible and infeasible solutions, average total time and the plan length are given in below graphs:

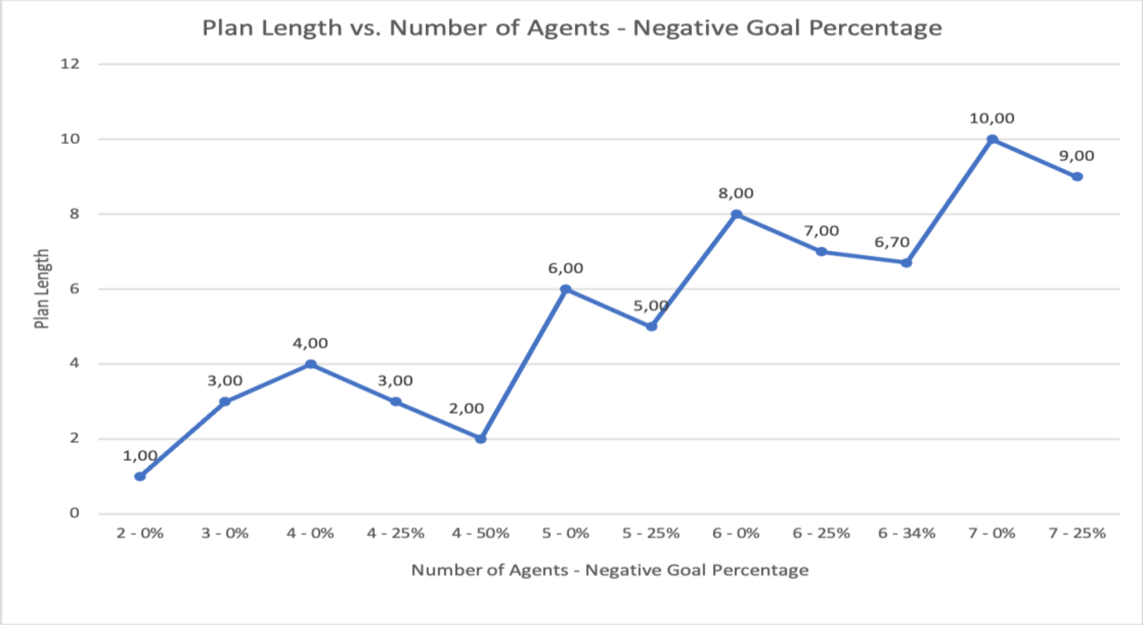

Feasible - Infeasible Solutions vs Agent Number - Negative Goal Percentage
Number of Feasible and Infeasible Solutions
80
70
60
50
40
30
20
10
0
2 - 0%
3 - 0%
4 - 0%
4 - 25%
4 - 50%
5 - 0%
5 - 25%
5 - 50%
6 - 0%
6 - 25%
6 - 34%
6 - 50%
7 - 0%
7 - 25%
7 - 50%
Infeasible 0 0 0 21 39 0 23 66 0 40 71 60 0 16 45
Feasible 5 5 10 4 1 10 2 0 10 10 3 0 5 8 0
Feasible
Infeasible
Number of Agents - Negative Goal Percentage


Average Total Time vs Number of Agents(n) - Negative Goal Percentage, for n = 4
0,0050
0,0045
0,0040
0,0035
0,0030
0,0025
0,0020
0,0015
0,0010
0,0005
0,0000
Total Average Time(s)
0,00363238
0,00263566
0,00431101
4 - 0%
4 - 25%
4 - 50%
Number of Agents - Negative Goal Percentage


Average Total Time vs Number of Agents(n) - Negative Goal Percentage, for n = 5
0,025
0,020
0,015
0,010
0,005
0,000
Average Total Time (s)
0,02049584
0,01398035
5 - 0%
5 - 25%
5 - 50%
Number of Agents - Negative Goal Percentage


Average Total Time vs Number of Agents(n) - Negative Goal Percentage, for n = 6
Total Average Time(s)
0,9
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0,0
0,7635419
0,14430027
0,01930817
6 - 0%
6 - 25%
6 - 34%
6 - 50%
Number of Agents - Negative Goal Percentage


Average Total Time vs Number of Agents(n) - Negative Goal Percentage, for n = 7
Total Average Time(s)
70
60
50
40
30
20
10
0
62,73362
6,5044925
7 - 0%
7 - 25%
7 - 50%
Number of Agents - Negative Goal Percentage


Plan Length vs. Number of Agents - Negative Goal Percentage
Plan Length
12
10
8
6
4
2
0
1,00
3,00
4,00
3,00
2,00
6,00
5,00
8,00
7,00
6,70
10,00
9,00
2 - 0%
3 - 0%
4 - 0%
4 - 25%
4 - 50%
5 - 0%
5 - 25%
6 - 0%
6 - 25%
6 - 34%
7 - 0%
7 - 25%
Number of Agents - Negative Goal Percentage

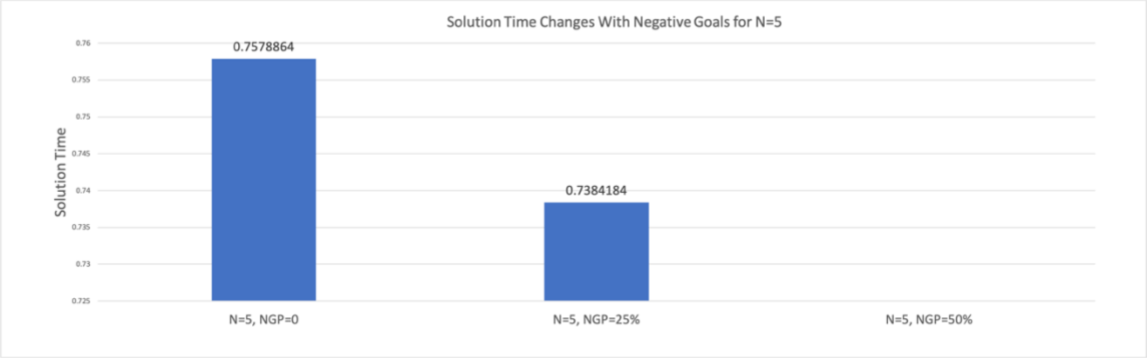
It is observed that average plan length is not an integer only when n=6 and p=34% because this case resulted in 3 feasible solutions, 2 of them had a plan length of 7 and the other one's plan length was 6. Therefore, the average plan length for this case is equal to 6,7.

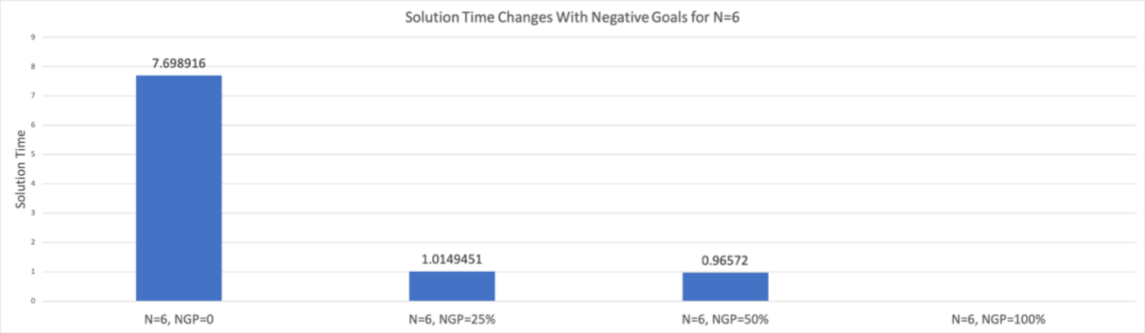
As a result, METIS1 worked very fast for the basic gossip problem for until n=8. For the gossip problem with negative goals, it was hard to generate random instances that resulted in feasible solutions, and it resulted in the need for seeds that were the instances of earlier tests.

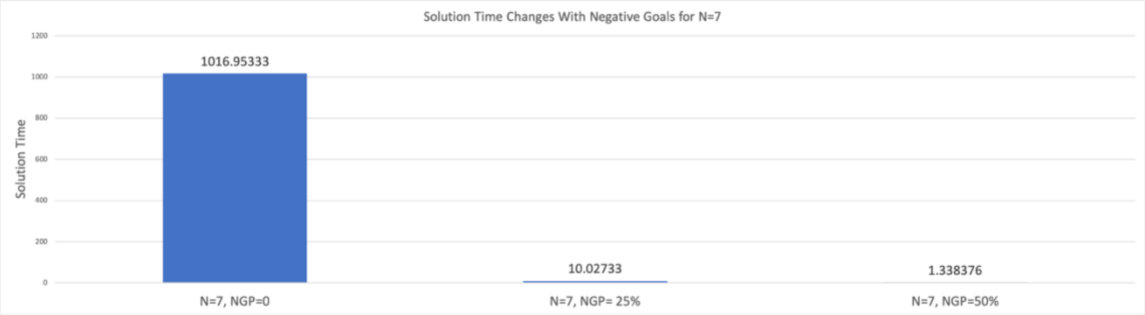
3. COMPLEMENTARY1 --- Zeki Oğuzhan İnce

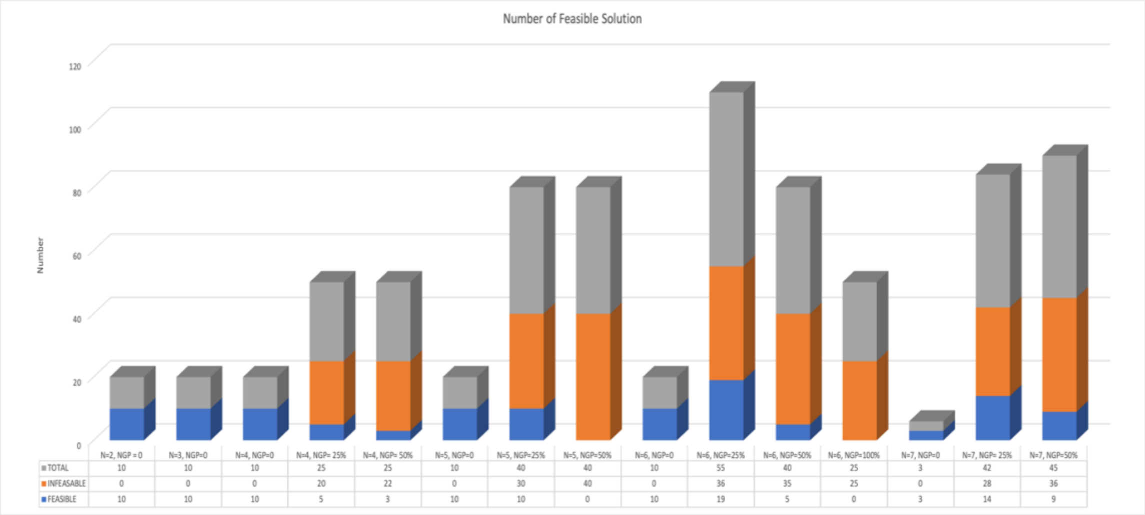
Firstly, the installation of the planner complementary1 is completed for the gossip server. After the installation, by using the gossip problem generator and the domain pddl which is written before is used to test necessary conditions. Basic gossip problem for the agent number 2,3,4,5…,8 is tested. Until the agent number equal to 7, 10 test case is done, and their average solution time is used for the obtained graphs/tables. For the agent number 7, 3 test case is done because of the solution time of the 7 agent is much bigger than the other case. By using those three-solution time, obtained average solution. Time for the 7-agent case. In the planner complementary1, there is no solution for the agent number 8 because it exceeds the time limit which is given as 1800s.

For the gossip problem with negative goals, the generator for the gossip problem with negative goal percentage is used in order to test those conditions. In the generator, there is random generator which generate by percentage for each agent. In other words, every percentage that given affects the negative goal for each agent. For example, for the agent number 4, when %25 percentage is used, each agent has one negative goal. If given percentage is %50, each agent has two negative goals. It changes to agent number. However, the generated seeds by the generator generally give infeasible solution for the planner. That's why, number of the feasible solution and the infeasible solution for each case is plotted as graph. On the other hand, in order to increase the feasible solution number, feasible solutions which found before is used in order to generate new seed. For instance, for agent number-4 with %25 negative goals, when the feasible seed found, while generating agent number 5 with %25 negative goals, that feasible seed found by the agent 4 is given as condition. This process continues for the next test cases in order to increase the feasible solution percentage. As a result, all time averages, feasible solutions and infeasible solutions are reported in the graphs given below:

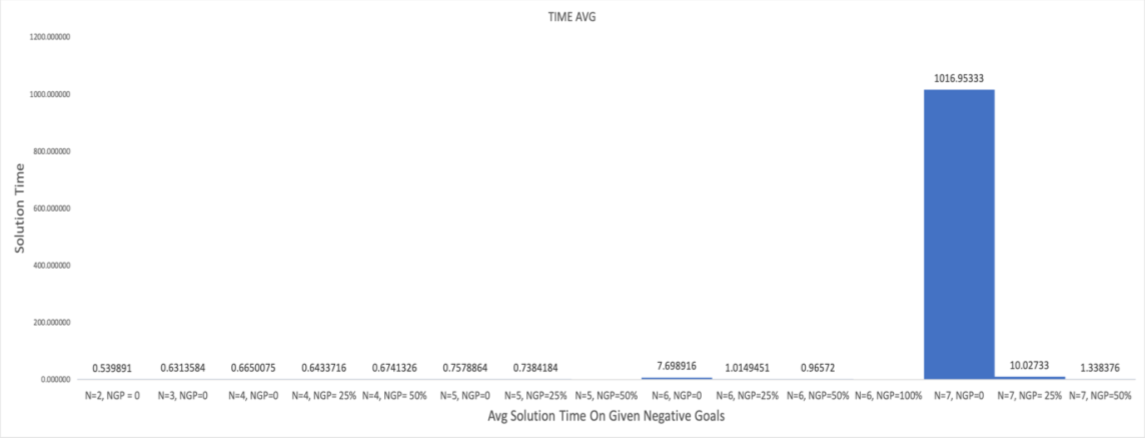



Solution Time Changes With Negative Goals for N=6


Solution Time Changes With Negative Goals for N=7


Number of Feasible Solution

	N=2, NGP = 0	N=3, NGP=0	N=4, NGP=0	N=4, NGP= 25%	N=4, NGP= 50%	N=5, NGP=0	N=5, NGP=25%	N=5, NGP=50%	N=6, NGP=0	N=6, NGP=25%	N=6, NGP=50%	N=6, NGP=100%	N=7, NGP=0	N=7, NGP= 25%	N=7, NGP=50%
■ TOTAL	10	10	10	25	25	10	40	40	10	55	40	25	3	42	45
■ INFEASABLE	0	0	0	20	22	0	30	40	0	36	35	25	0	28	36
■ FEASIBLE	10	10	10	5	3	10	10	0	10	19	5	0	3	14	9


TIME AVG

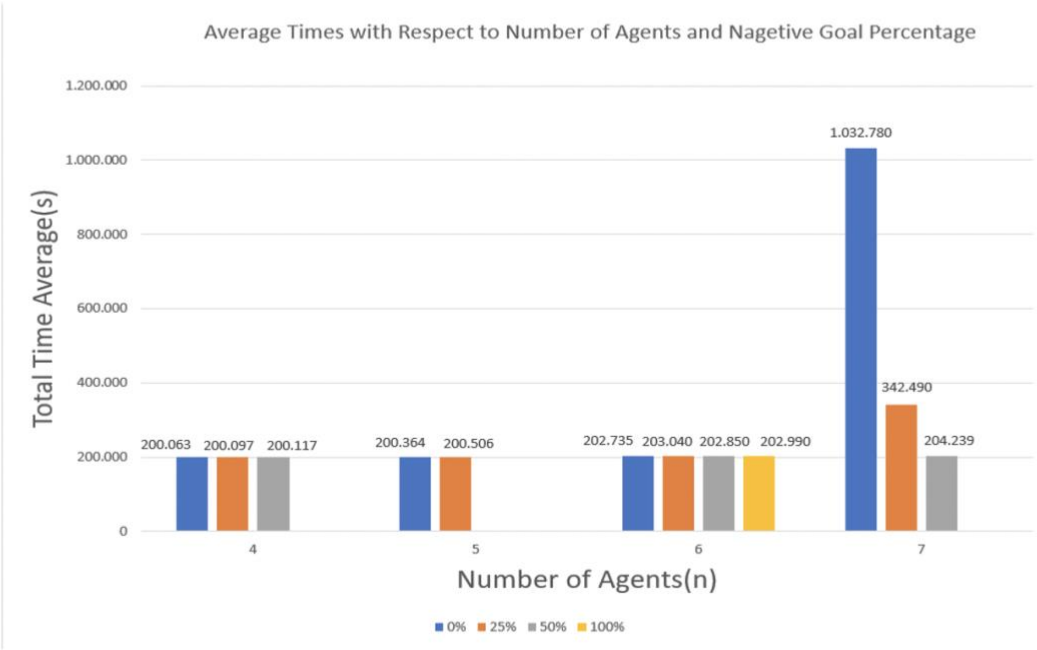
4. SCORPION --- Taygun Özcan

After the installation of Scorpion planner to my directory, I tested given conditions by using domain pddl that we already have and problem pddl that generated by problem generator in Haşim's directory. It was harder to test many times with scorpion planner than the other planners because the minimum time to search for any problem was 200 seconds. I've tested basic problem for n= 4, 5, 6, 7 and problem solved for each of them. For n bigger than 7, I could not get solution.

Testing problems with agents that have negative goals didn't give accurate solutions as much as basic problem in terms of feasible solutions. Although, I have tasted each problem 10 times I could only get solutions for 4 1(%50) and 5 1(%25). So, I had test problems that are generated by hand to get a feasible solution for other conditions.

For n= 4, 5, 6; total times were close to each other. For n= 4 it was around 200,1s, for n=5 it was around 200,6s and for n=6 it was about 203s. But for n=7 it was bigger than the others.

At the end, I could not get solution for 5 2(%50), 5 3(%100) and 7 5(%100). But I was able to get solutions for other problems. And the graph below shows total times for each problem.



Test Results with Increasing Epistemic Depth without Negative Goals

After the tests for depth 1, the team members were asked to focus on turning the gossip problem into an epistemic knowledge problem. In order to do that and to understand epistemic knowledge in general, two articles were assigned, and each team member was asked to work individually in order to create a new domain file for epistemic depth of 2 in PDDL and use the planners they were assigned before to run their tests. Reports of each member and their tests results are given below:

1. Planning-PDBs --- Haşim Sait Göktan

Forall statements improve readability in domain files. I started with domain files generated
using Faustine Maffre’s GossipProblem-PDDL generator:
https://github.com/FaustineMaffre/GossipProblem-PDDL-generator

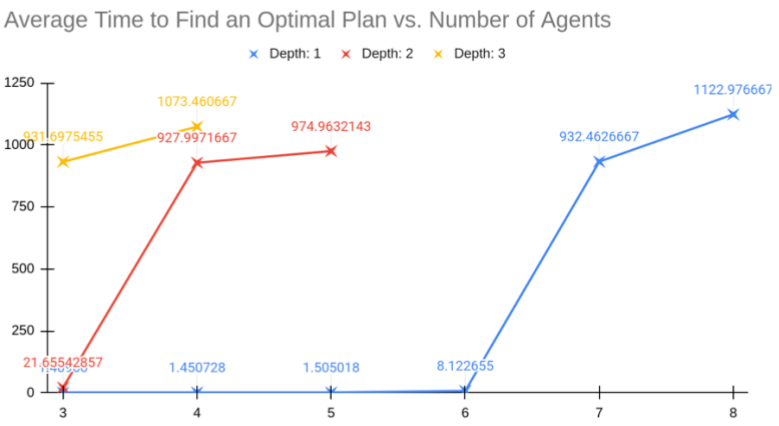
It is easy to see for every n lines, only the secret changes. We can replace secret terms with ?s and add (forall (?s) to the beginning.

Epistemic depth 2’s domain files have n additional lines where the only change is the agent term. We can replace the agent term with ?a and add (forall (?a) to the beginning. These nested forall statements allow us to use a single domain file for any n at epistemic depth 2. First line is for depth 1 knowledge ie. the effects that will take place when agent1 knows secret2. Second line is for depth 2 knowledge.

Epistemic depth 3’s domain files have n*(n-1) additional lines where the second agent term changes each line and the first agent changes every n-1 lines. This is due to the fact that knows-3 atoms can’t be “agent$_i$ knows agent$_j$ knows agent$_k$ knows secret$_l$” where i=j or j=k. i ≠ j is included in my forall domain file using (forall (?ag1) (forall (?ag2) (and (not (= ?ag1 ?ag2)). Same applies to knows-2, which is why every line after the first one starts with (and (not (= ?i ?a)) (not (= ?j ?a)).

Domain files for higher epistemic depths can be generated using this pattern. I avoided using types since most of the planners do not support types specification. I verified the correctness of my domain files by comparing the optimal plans for instances with different number of agents and depth values to the optimal plans for instances generated by Faustine Maffre’s generator with the same number of agents and depth values.

Final version of the generator is publicly available on https://github.com/hasimsait/gossip_problem_generator. To run tests with any number of agents and depth, run the command “/path/to/problem_generator.py d n p &&\singularity run -C -H
/path/to/my/rundir/path/to/my/planner/container/current/working/directory/domain_<d>.pddl /current/working/directory/problem_<d>_<n>_<p>.pddl /my/rundir/sas_plan” where d is depth, n is number of agents, p is percentage of uniformly distributed random negative goals per agent. Corresponding graph is given below:



Results of the tests with Planning-PDBs shows that as the depth increases, the average time to find an optimal solution increase. For n and d values that are too low, an optimal solution can be found during the initial search without utilizing any of the planner's features, providing no useful information regarding the planner's performance and some information on the complexity of the problem. For depth 3, finding an optimal solution for 5 agents takes more than 11000 seconds and requires more than 60 GB of memory therefore I could not run that test.

All tests are performed on Linux node (Ubuntu 14.04) with 2 x Intel Xeon E5-2665 @ 2.40GHz, 64GB total memory using singularity containers as instructed on https://ipc2018-classical.bitbucket.io.

2. METIS1 --- Berkin Karaçam

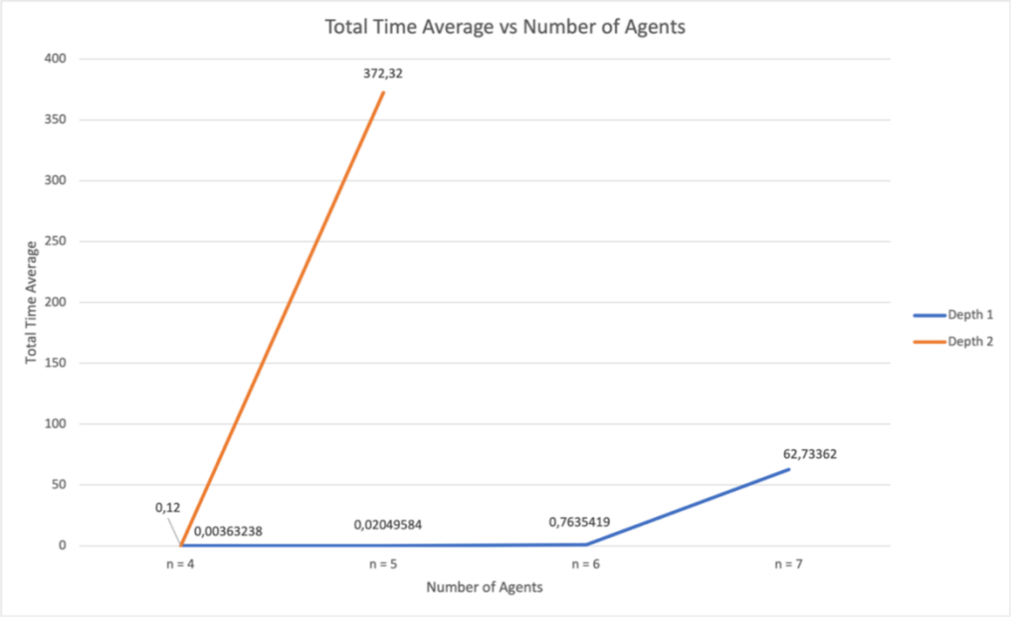
We were asked to create a domain file for the representation of the gossip problem as epistemic knowledge problem, using epistemic depth 2. Our base was the PDDL generator: https://github.com/FaustineMaffre/GossipProblem-PDDL-generator.

This generator creates the domain file without using forall statements. In order to simplify the file and to make it easier to understand, I changed the action description. Using forall statements also helps us by enabling us to create only one domain file for given depth, we don't have to create different domain files for different number of agents.

The first change to the domain file was to add another predicate, (KW ?a1 ?a2 ?s). When this predicate is true, it shows that agent a1 knows that agent a2 knows the secret s. This is simply the way we add the epistemic depth to our domain.

The call action takes two parameters, agent1 and agent2. For the basic gossip problem, we used one forall loop for all of the secrets and if any of these secrets were known by one of the agents, the other one would learn this secret. First of all, inside this forall statement I added two more effects: (KW ?agent1 ?agent2 ?s) (KW ?agent2 ?agent1 ?s). The meaning of this addition is, if a secret is known by any of the two agents, in addition to making sure that they will both know the secret from now on, they will also both know that the other agent knows this secret too. The main difference in the domain file is checking about other agents who are not a part of the call. In epistemic depth 2, we also have to make sure that if one of the two agents and another agent who is not in the call both know a secret, and if the agent in the call also knows that this other agent knows the secret, this information will be shared with the other agent in the call. For example, agent 1 and agent 3 know secret s and agent 1 knows that agent 3 knows this secret s. After the call between agent 1 and agent 2, agent 2 will learn both the secret s and the fact that agent 3 knows secret s. In order to achieve that in the domain file, nested forall statements were added to the action description. The outer loop checks the agents, and the inner loop checks the secrets. For all of the secrets that are known by one of the agents outside the call, when one of the agents in the call knows this secret and the fact that the agent outside the call knows this secret, this information is shared with the other agent in the call that did not know about the secret.

After changing the domain file, I tested this new domain file for different number of agents. I started from 4 agents and tested 5 agents as well. For 6 agents, METIS1 planner could not find a solution for the given time limit. Finding the solution took 0.12 seconds for 4 agents and took 372.32 seconds for 5 agents. The average time vs number of agents graph for both epistemic depth 1 and 2 is given below:



3. COMPLEMENTARY1 --- Zeki Oğuzhan İnce

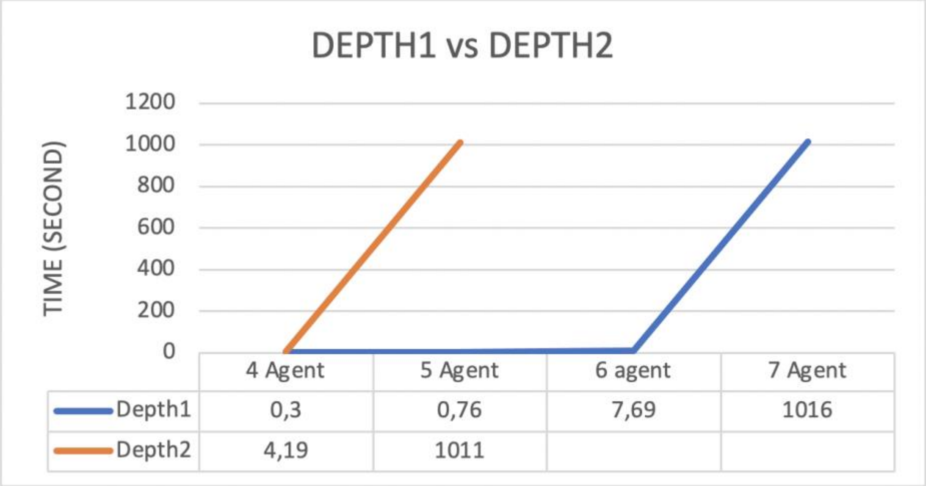
We were responsible for the new topic about the Gossip Problem which based on the knows-whether approximation. First three weeks, we read and analyzed the documents about the depth-based gossip problem. By using the generator by the Faustine Maffre' s generator, https://github.com/FaustineMaffre/GossipProblem-PDDL-generator.

I generate domain and problem pddls for the depth-2. However, our domain files was quite complex. We were responsible for the generate domain file by using the forall approximation like we use last semester in our domain file.

Domain file that we generate by using the generator includes unnecessary lines which we can obtain those in the forall loop. Also, all the domain files need to be different without forall because we had to define actions for each number of agents again. Including loops in the action makes the code more understandable and readable but it was really hard to convert it to forall version for me. Firstly, I noticed that I need to add new predicate as "KNOWS_WHETHER ?a1 ?a2 ?s" in the domain file which mean that a1 knows that a2 knows the secret s. The main effect of this predicate in the action is that after each call, each agent will learn both the secrets of the other agent and the information that the other agent knows these secrets. Also, for all of the other agents outside the call, we need to check if there is a shared secret between one of these agents and one of them inside the call. If one of the agents in the call and an outside agent both know the same secret and if the agent in the call also knows that the secret is known by the outside agent, this information will be shared with the other agent on the call. From now on, the other agent in the call will know that the outside agent knows that secret. In order to achieve these, I made an addition to the effect of the forall statement that was implemented before and created a nested forall statement to check for each agent outside the call and each of the secrets.

After the implementation of domain pddl, I started to test in the remote server which used in the last semester. In the complemtary1, first I got an warning and it did not work because the pddl files includes types agent and secret. Planner does not support the types in the pddl and that's why, I had to delete them from the files. After that, I put the time limit 1800s and memory 8388608 KB for the tests. I started to the tests with the 4 agent. It takes more time compared to the depth1. The total time is 4.19s for it and it uses 11112232 KB memory. After that, I tested the 5 agent problem and it takes much more time compared to the depth1. It takes 1011s and it used 3002380 KB memory. For the 6 agent, It reached to the memory limit and can't find a solution. Also, it was so close to the time limit, it takes approximately 30 minutes and before the time ends up, memory is reached.

Corresponding test results are given in the graph below:



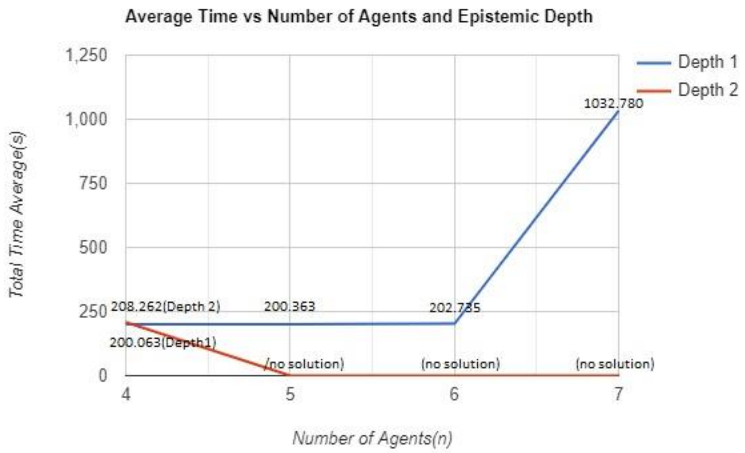
	4 Agent	5 Agent	6 agent	7 Agent
Depth1	0,3	0,76	7,69	1016
Depth2	4,19	1011		

4. SCORPION --- Taygun Özcan

In this semester, initially we read articles about epistemic knowledge problem to understand the problem and learn how to implement in gossip problem. We had Faustine Maffre's (https://github.com/FaustineMaffre/GossipProblem-PDDL-generator) generator as an example. And we are asked to modify the given 2-depth domain pddl file by adding forall statement. Since epistemic depth is 2, I added (S-2 ?i ?j ?s) to predicate part. This predicate indicates "agent i knows that agent j knows secret s". For epistemic depth 2, when an agent calls another agent, agents will know not only the secrets that each other knows but also the knowledge of which secrets they know. Also, if both agents on the call knows a secret and one of them knows that any other agent also know this secret, other one also learn(know) that other agent knows this secret. Action part checks these conditions by forall statements.

I used Scorpion planner as I did last semester. This time scorpion was not able to find solution in 30mintes and I get error for epistemic depth 2, number of agents are 5, 6, 7. I only got a solution for epistemic depth 2, number of agents is 4. But it took pretty long, 208.262s.

As a conclusion scorpion planner is not able to solve the problem when epistemic depth is more than 1. Corresponding graph is given below:



Experiments on Negative Goals in Higher Epistemic Depth Gossip Problems

Generating negative goals using disconnected networks:

When agent i is isolated from the network, the goals KNOWS(x,s) where x = i xor s = i in the goal state are turned to negative goals since no other agent can learn i's secret and i can't learn any other agent's secret since no other agent is reachable from agent i. When x=i and s=i, the goal is positive since that goal is positive in the initial state. At depth 2, the predicate KNOWS-2 gets the same treatment, the goals KNOWS-2(x,y,s) where x=i or y=i or s=i become negative goals. When isolating 2 agents i and j from the rest of the network, same method applies but the goals KNOWS(x,s) where x=j, s=i and x=i, s=j are left positive since agent i is reachable from agent j. At depth 2, KNOWS-2(x,y,s) goals where x=i or j ,y=i or j, s=i or j are positive goals, rest are negative as if both agent i and agent j are isolated from the rest of the network. Which is actually the case since we're adding the edge between agents i and j after isolating both of them from the network.

To summarize, in order to disconnect x agents from the rest of the network while keeping the edges between those agents, make all goals mentioning those agents' negative goals except the ones that are between the disconnected agents themselves. The instances that are included in the repository are named accordingly, when one agent is separated from the network, the name of the file ends with sepone, when the sub-graph of 2 agents is separated from the network, the file is named septwo, when 2 sub-graphs of 2 agents is separated, the file is named septwotwo.

1. Planning-PDBs --- Haşim Sait Göktan

   In order to generate feasible problem instances for higher epistemic depths, after thousands of unsuccessful attempts and placing an upper bound on plan length using fluents; which had a negative effect on the performance (implementation details below), I used disconnected graphs. The domain description used in this part of the research does not contain the cancall precondition, the network is still connected and complete. But the goal state is the goal state of a disconnected graph. When you disconnect an agent from the gossip network, you must remove the goals for that agent's secret since the removed agent is no longer reachable from other agents. As you separate the gossip network into smaller and smaller disconnected subnetworks, the number of negative goals also increase.

Placing an upper bound on plan length in PDDL:

Numeric fluents are not supported in many planners. To overcome this limitation, one needs to introduce the fluent as predicates. Since a predicate cannot cover all possible states of a fluent, you must convert the entire range of possible values the fluent can take to predicates, each representing a single state.

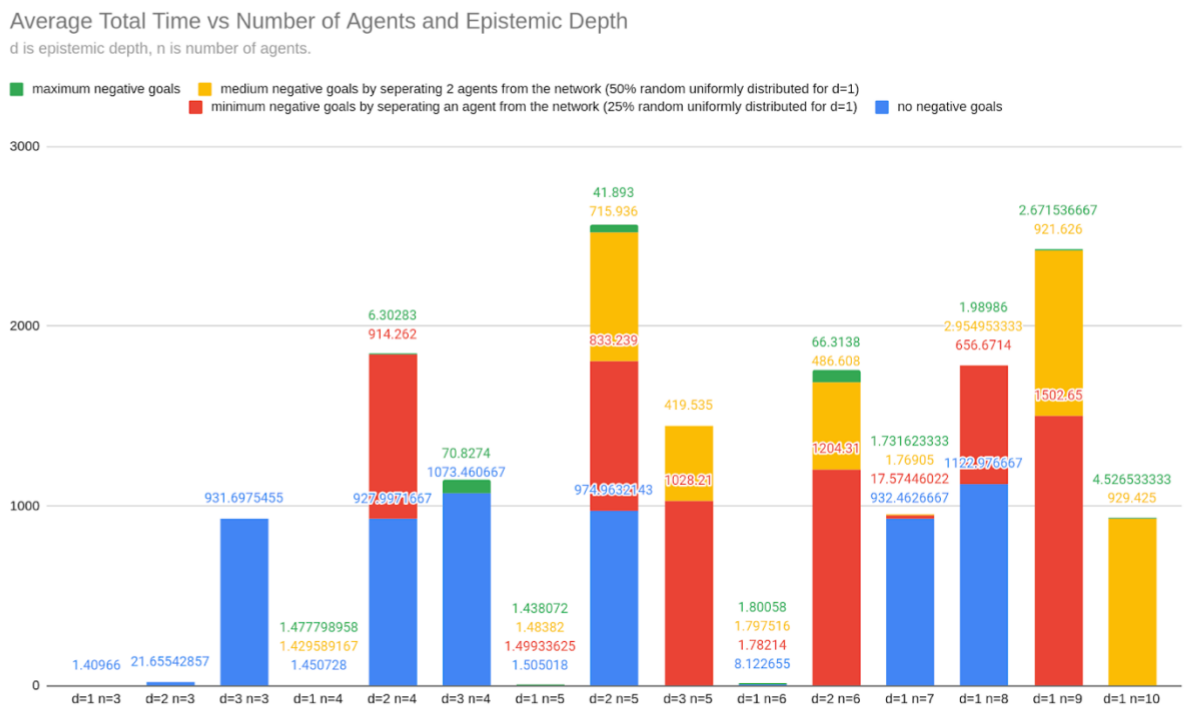
```
(define (domain domainGossip)
(:requirements :adl)
(:predicates
    (KNOWS ?a ?s)
    (next ?m ?n)
    (at ?m)
)
(:action call          :parameters (?a1 ?a2 ?m ?n)
    :precondition (and (next ?m ?n) (at ?m))
    :effect (and (forall (?s)
            (when
                (or
                (KNOWS ?a1 ?s) (KNOWS ?a2 ?s)
                )
                (and
                (KNOWS ?a1 ?s) (KNOWS ?a2 ?s)
                )
                ))
            ;;increase total cost
            (not (at ?m)) (at ?n)
            )

))
```

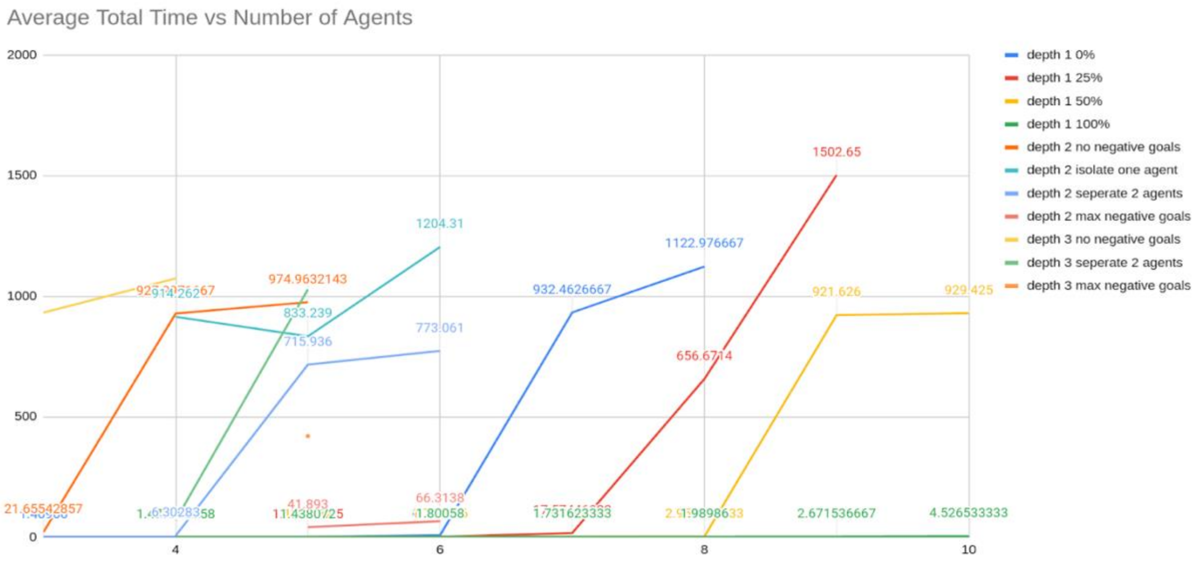
Then in the problem definition, for every step in the entire range of your numerical fluent, create an object. Then for each of those objects, add (next mi mj) where j is the next step from i. The problem files can be generated using commit 079c123f8f8d29d8689852936a05ad12f153d505 in https://github.com/hasimsait/gossip_problem_generator. With this domain, no plan for any instance could be found within 2 hours using Planning PDBS. All tests below are performed using the old domain on problem instances generated with the method described above.

For the method with disconnected graphs described above, the minimum number of negative goals for any network is isolating one agent; maximum can be achieved by separating the network into groups of 2 agents. The average time of every instance is given under Appendix A.

These results are in the graphs below:



Average running time of Planning PDBS on various gossip problem instances



Average running time of Planning PDBS on various gossip problem instances

The instances with negative goals on higher depth take longer to solve than their lower epistemic depth, same percentage of negative goals counterparts. As the percentage of negative goals increases after some point, the time required to find an optimal plan decreases at higher epistemic depths, same as epistemic depth one. When a problem instance contains only few negative goals, it is a harder problem than no negative goals since at epistemic depth 3, when there are no negative goals, the problem can be solved within 1073s for 4 agents, but when we isolate an agent from these agents, the problem requires more than 32GBs of memory and more than 2 hours. Isolating an agent with negative goals is not the same thing as deleting that agent in terms of problem complexity since the problem instance with 3 agents at depth 3 can be solved within 931 seconds with less than 3GBs of memory.

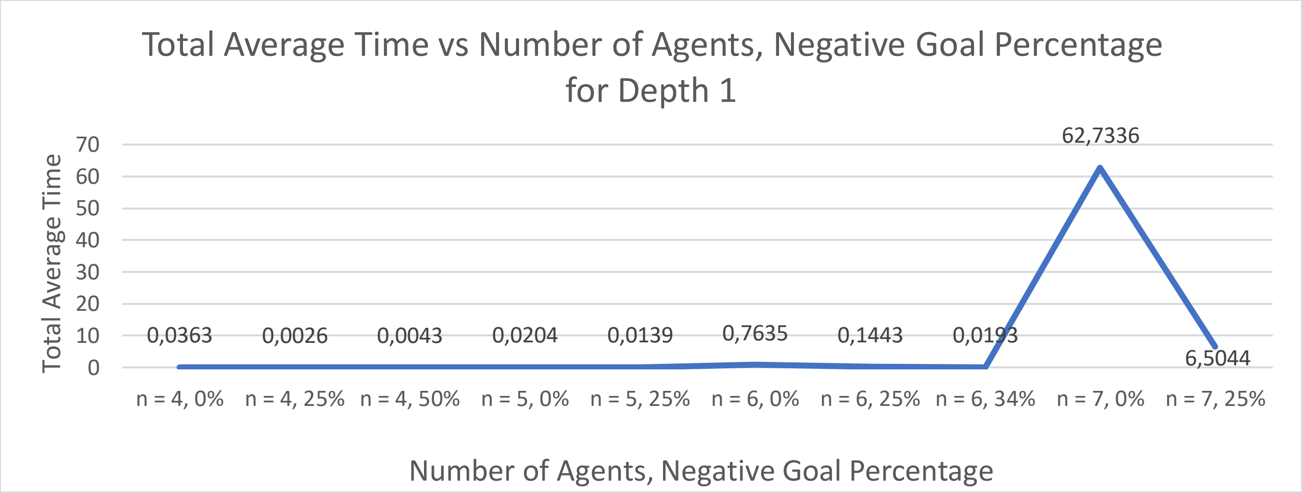
2. METIS1 --- Berkin Karaçam

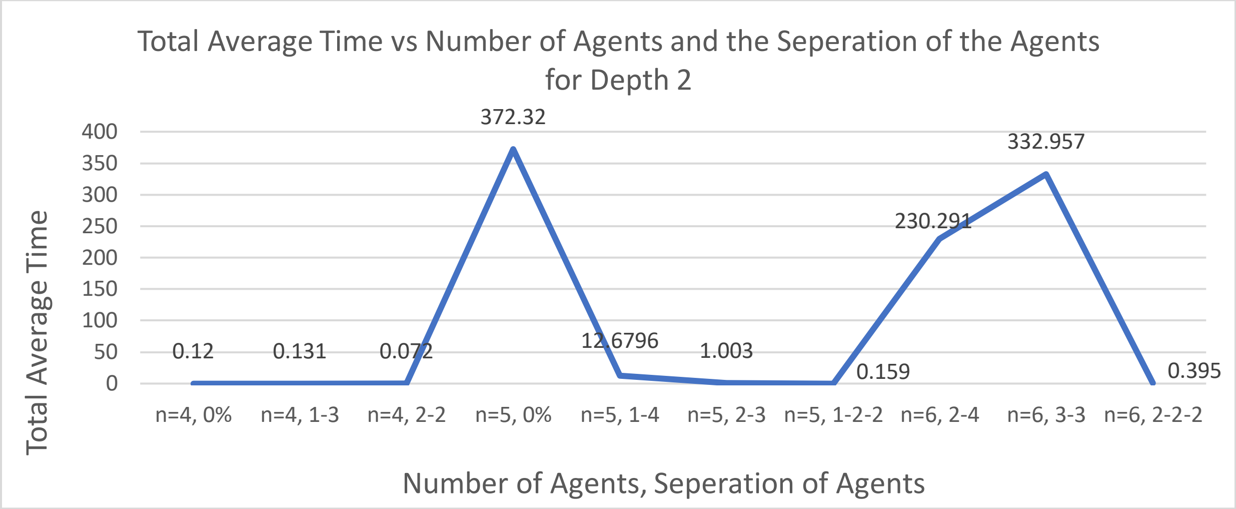
For epistemic depth 2, it is very hard to create feasible instances with negative goals. After a lot of attempts on this, I was unable to create any feasible instances for the planner to solve. Then, we decided to change our approach. We thought that, disconnecting some number of agents would help us with the problem. While disconnecting an agent from the graph, you have to remove the goals that include the given agent from the domain description. This will also mean to create more negative goals for the instances. For n=4,5 and 6, we created these instances manually, instead of using a generator.

Separating one agent from the graph resulted in fewest number of negative goals and increasing the number of disconnected subgraphs maximized the number of negative goals. For example, when n=6, separating one agent minimized the negative goal number and separating the graph as 3 subgraphs that have 2 agents maximized this number.

METIS1 worked very fast again like it did while experimenting for epistemic depth 1. However, it was unable to find a solution for n=6, when 1 agent is separated from the graph. As a result, METIS1 planner works very fast, but is usually unable to find a solution in given time and memory constraints.

The total average time graphs for given number of agents and negative goal percentage for depth 1, and for given number of agents and the separation of agents for depth 2 are given below:

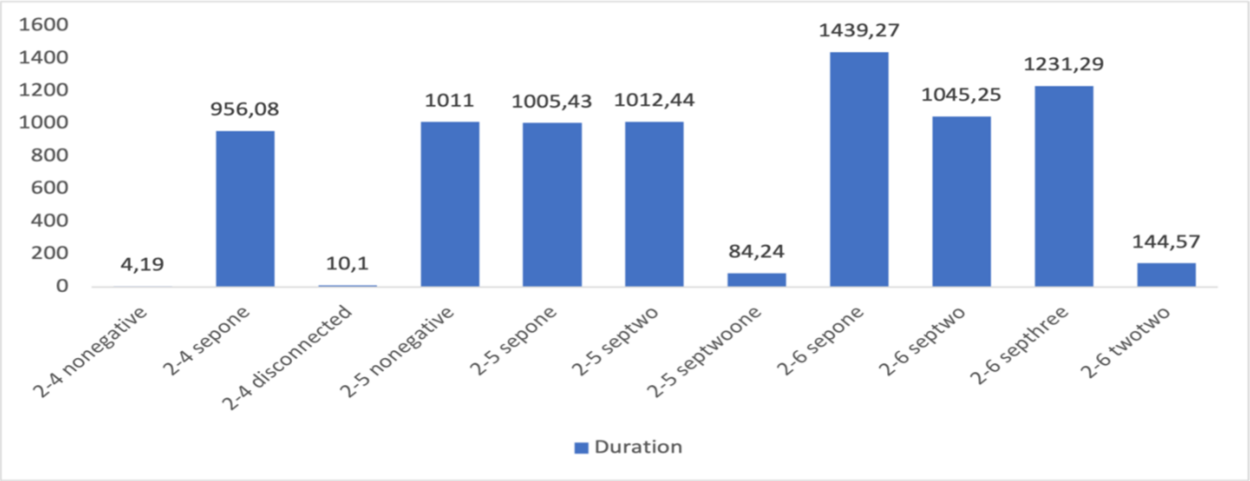




3. COMPLEMENTARY1 --- Zeki Oğuzhan İnce

In the experiments of the depth2 tests, I started with using of generator in order to obtain feasible cases. However, although I tried to reach feasible solutions many times, I couldn't get any feasible solution. As a solution, I decided to use disconnected graphs as a negative goal. As disconnected agent number increases, the negative goal percentage increase. With the disconnected graphs, the given number of agents are disconnected from the other agents, so they can't be reachable by the other agents. That solution helps me to obtain results and they are shown in the graph.

In the result of experiment, when there are 4 agents in the problem, without negative goals is much faster than negative goals. However, when we come to the 5 agents, negative goal percentage doesn't change so much time for the problems except 2-5 separated two-one-one. And also, without negative goals complementary1 is not able to solve problem in given limited time. On the other hand, when there are negative goals on the problem, planner can solve the problem until 6 agents. Detailed times of the experiment shown in the graph.



4. SCORPION --- Taygun Özcan

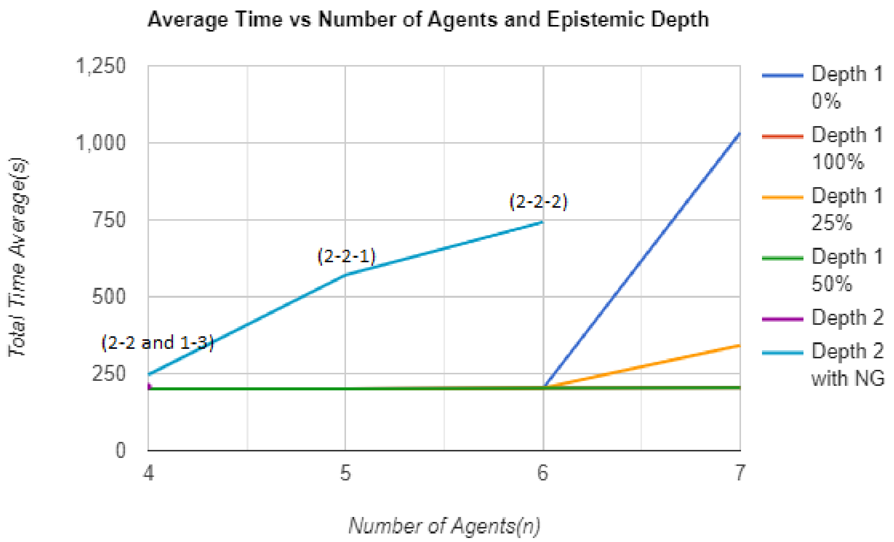
We used disconnected graphs during depth 2 experiments to find feasible solutions.

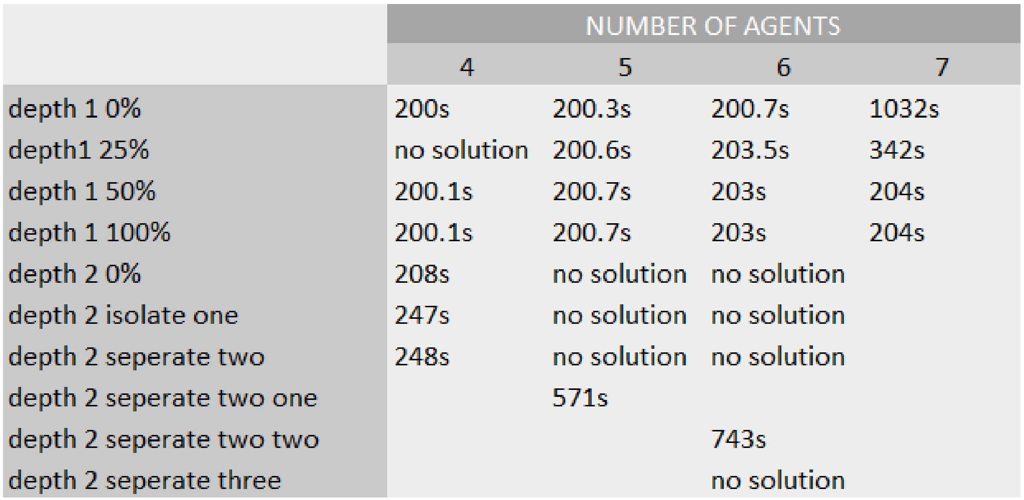
In experiments by using Scorpion, when depth = 2 without negative goals, solution can be found only for n= 4. On the other hand, when depth = 2 with negative goals, solution can be found for n= 4, 5, 6.

For n= 4 without negative goals, total time to find a solution was 208 seconds. For n= 4 with negative goals total time to find solution was 247 seconds (for 1-3) and 248 seconds (for 2-2).

For n= 5 with negative goals, only one solution can be found which took 571 seconds to find (for 2-2-1).

For n= 6 with negative goals, again only one solution can be found, and it took 743 seconds to find (for 2-2-2).





	NUMBER OF AGENTS			
	4	5	6	7
depth 1 0%	200s	200.3s	200.7s	1032s
depth1 25%	no solution	200.6s	203.5s	342s
depth 1 50%	200.1s	200.7s	203s	204s
depth 1 100%	200.1s	200.7s	203s	204s
depth 2 0%	208s	no solution	no solution	
depth 2 isolate one	247s	no solution	no solution	
depth 2 seperate two	248s	no solution	no solution	
depth 2 seperate two one		571s		
depth 2 seperate two two			743s	
depth 2 seperate three			no solution	

Experiments on Removing Connections in Gossip Problems Using Planning PDBS

A gossip network can be converted to a graph. The problem is feasible if and only if every node of the graph is reachable from every other node. The number of edges necessary is n-1. In a complete network, there are n-1+n-2+n-3+..1+0 = n*(n-1)/2 edges. Therefore there are (n/2)*(n-1)-(n-1)=(n-2)(n-1)/2 redundant edges in any complete network.

On the complete network of 3 agents, which can be seen in Appendix H, one of the edges can be removed and agent C would still be reachable from agent B via agent A. 1*2/2= 1. 1 redundant edge.

On a complete network of 4 agents, in Appendix I, the same case applies, there are 2*3/2=3 redundant edges.

In this part of the research, without negative goals, edges will be removed from complete gossip networks and how the number of edges affects planner performance will be studied.

The problem instances and the domain description for incomplete networks can be accessed from the incomplete_graphs directory under the incomplete_graph branch of https://github.com/hasimsait/gossip_problem_generator. The plan length for any instance is either 2n-3 or 2n-4, depending on the existence of a loop of length 4.

The results below are as expected, as the number of edges decreases, after some point, the precondition decreases the number of possible states that can be reached from a state therefore the time it takes to find an optimal plan decreases. The number of expanded states decreases after some of the edges are removed. Same trend as negative goals but the steps are eliminated before their effects are compared with the goals of the state. The higher depths require too much memory even with many edges removed. The plan lengths found proves that the optimal plan length at epistemic depth 1 for any gossip network is less than or equal to 2n-3.

The results for epistemic depth 1 are included in Appendix B. We can observe that the plan length is always less than or equal to 2n-3 for epistemic depth one. We can separate the average total time to achieve the table included in Appendix C, which can be visualized with a line chart as Appendix D.

Due to the high memory requirements of higher epistemic depths, the results are limited to less than 4 agents at epistemic depth 2. At epistemic depth 3, no problem instances except 3 and 4 agents with no edges removed, can be solved within memory limitations with Planning PDBS.

The results for epistemic depth 2 and a line graph for that visualizes these results are included in Appendix E and Appendix F respectively.

5. IMPACT

The project was mainly based on experimenting. We implemented different algorithms and used different AI methods in order to solve the gossip problem and its variations and performed a lot of different experiments using different planners. For these reasons, we reached to different results while using different methods. These results did not contribute to any commercial or entrepreneurial aspects of the project.

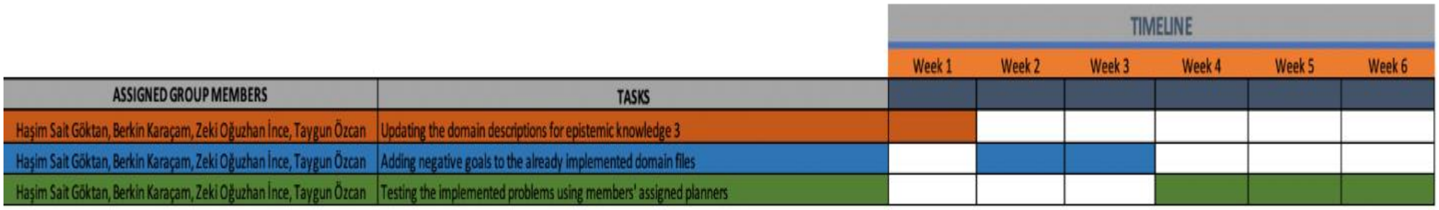
Furthermore, since we only used a PDDL generator as source code and referenced it, we do not have any Freedom-to-Use issues.

6. ETHICAL ISSUES

Since we will not reach any kind of personal information during the implementation of this project, there are not any ethical issues for this project. The thing we had to be careful about was referencing all of the external sources we are using.

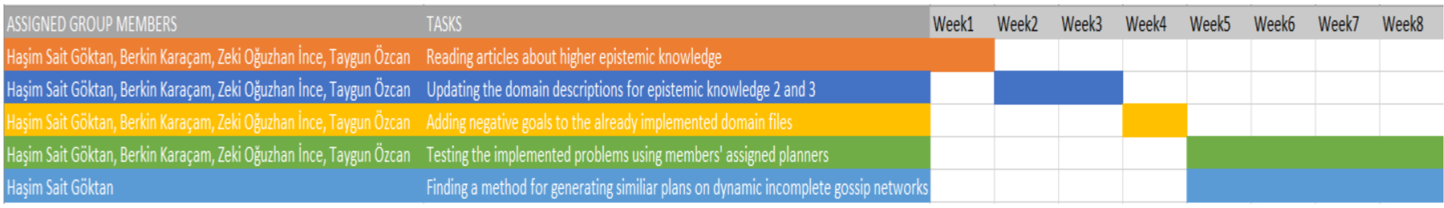
7. PROJECT MANAGEMENT

Initial project plan of the second term of the project:



		TIMELINE					
		Week 1	Week 2	Week 3	Week 4	Week 5	Week 6
ASSIGNED GROUP MEMBERS	TASKS						
Haşim Sait Göktan, Berkin Karaçam, Zeki Oğuzhan İnce, Taygun Özcan	Updating the domain descriptions for epistemic knowledge 3	🟧					
Haşim Sait Göktan, Berkin Karaçam, Zeki Oğuzhan İnce, Taygun Özcan	Adding negative goals to the already implemented domain files		🟦	🟦			
Haşim Sait Göktan, Berkin Karaçam, Zeki Oğuzhan İnce, Taygun Özcan	Testing the implemented problems using members' assigned planners				🟩	🟩	🟩

Final project plan of the second term:



ASSIGNED GROUP MEMBERS	TASKS	Week1	Week2	Week3	Week4	Week5	Week6	Week7	Week8
Haşim Sait Göktan, Berkin Karaçam, Zeki Oğuzhan İnce, Taygun Özcan	Reading articles about higher epistemic knowledge	🟧							
Haşim Sait Göktan, Berkin Karaçam, Zeki Oğuzhan İnce, Taygun Özcan	Updating the domain descriptions for epistemic knowledge 2 and 3		🟦	🟦					
Haşim Sait Göktan, Berkin Karaçam, Zeki Oğuzhan İnce, Taygun Özcan	Adding negative goals to the already implemented domain files				🟨				
Haşim Sait Göktan, Berkin Karaçam, Zeki Oğuzhan İnce, Taygun Özcan	Testing the implemented problems using members' assigned planners					🟩	🟩	🟩	🟩
Haşim Sait Göktan	Finding a method for generating similiar plans on dynamic incomplete gossip networks					🟦	🟦	🟦	🟦

During the implementation, since we did not consider learning how to update the domain description for epistemic knowledge 2 and 3 in the initial project plan, we had to include a new task "Reading articles about higher epistemic knowledge" in the final project plan.

We also encountered unexpected problems. During the implementation of "Testing the implemented problems using members' assigned planners" task, memory of Turing was full. In the next meeting we informed our supervisor and got solution recommendations from her, so stated task took longer than we expected.

According to what we experienced in this project, we understand that effective communication with group members is crucial in order to find solutions to encountered problems and to move faster by helping each other. On the other hand, we also learnt that when we start our task last days of the deadline, it can be hard to handle unexpected problems. So, it is always better to start earlier to see possible problems we can encounter and determine how complex is the task.

8. CONCLUSION AND FUTURE WORK

In the project, we were responsible from testing with different planners. Each member obtained some results from the tests, and it showed that for each planner; time, memory usage, optimal plan length was different than expected.

Most Important Findings

Discussion about the Test Results for Epistemic Depth 1

First of all, for the basic gossip problem without any negative goals, it was clear that METIS1 was the fastest planner and SCORPION was the slowest one compared to others. However, for the tests in which the number of agents were 8 and higher, Planning-PDBs was the only planner that could successfully complete the tests.

Secondly, since each group member used the same problem generator that used random generation for negative goals, finding feasible solutions to the gossip problem with negative goals was hard for everyone. Especially as the number of agents increased, the probability of having a feasible instance with negative goals decreased. Even though a lot of tests were run, the number of feasible solutions remained very low. Therefore, the group members decided to use the instances that were generated in the earlier steps as the seeds for their other tests. The usage of these seeds resulted in more feasible solutions in less tries. However, it did not result in a drastic change.

The effect of adding negative goals on total time was surprising for the group members. For the planner SCORPION, the execution time did not change very much after adding negative goals. However, for Planning PDBs, METIS1 and COMPLEMENTARY1, adding negative goals resulted in decreased time. Especially as the number of agents increased, the decrease in execution time became very large for these planners.

As a result, even though METIS1 was clearly the fastest one out of these four planners, Planning-PDBs seemed to be more suitable for increasing number of agents and negative goals.

Discussion about the Test Results for Epistemic Depth 2

At epistemic depth 2 and 3, with our method of generating negative goals, as the number of negative goals increase, the size of the disconnected graphs decreases, simplifying the problems. Since the search space of the problem and the solution length decreases, the time necessary for the planners to find a solution decrease.

When we compared the planners with each other, we saw that METIS1 was the fastest planner, and the SCORPION was the slowest. However, neither of these planners were useful to find plans for more complex graphs. As the number of agents increased, Planning-PDBs and COMPLEMENTARY1 found plans that METIS1 and SCORPION could not. For example, we were unable to find solutions when number of agents were 6 and we separated 1 one of the agents with METIS1 and SCORPION, but Planning-PDBs and COMPLEMENTARY1 gave us solutions where the average total time took more than 1000 seconds.

Next Step and Future Work

Discussion about the Test Results for Epistemic Depth 3

Trying to create instances with negative goals for epistemic depth 3 can be done in the future. The results may be compared for each of the planners, may be considered as the next step of the project.

9. APPENDIX

APPENDIX A

	3	4	5	6	7	8	9	10
depth 1 0%	1.40966	1.450728	1.505018	8.122655	932.4626667	1122.976667	MEM	MEM
depth 1 25%	INF	INF	1.49933625	1.78214	17.57446022	656.6714	1502.65	MEM
depth 1 50%		1.429589	1.48382	1.79751	1.76905	2.95495333	921.626	929.425
depth 1 100%		1.4777989	1.438072	1.80058	1.731623333	1.98986	2.6715	4.52653
depth 2 0%	21.6554	927.99716	974.96321	MEM	MEM	MEM	MEM	MEM
depth 2 isolate one agent		914.262	833.239	1204.31	MEM	MEM	MEM	MEM
depth 2 seperate 2 agents		6.30283	715.936	773.061	MEM	MEM	MEM	MEM
depth 2 100%		INF	41.893	66.3138	MEM	MEM	MEM	MEM
depth 3 0%	931.6975	1073.4606	MEM	MEM	MEM	MEM	MEM	MEM
depth 3 isolate one agent		MEM	MEM	MEM	MEM	MEM	MEM	MEM
depth 3 isolate 2 agents		70.8274	1028.21	MEM	MEM	MEM	MEM	MEM
depth 3 3 disconnected graphs		INF	419.535	MEM	MEM	MEM	MEM	MEM

MEM= The instance requires more memory than available (32GBs)
INF=Infeasible, no such instance can be generated.

APPENDIX B

n, #removed edges	Average Total Time	Peak Memory	Plan Length
3 0	1.41285	815088	3
3 1	1.40112	815084	3
4 0	1.42248	938140	4
4 1	1.48779	938168	4
4 2	1.49373	938100	5

4 3	1.49162	938100	5
5 0	1.6133	1032852	6
5 1	1.65705	1034272	6
5 2	1.60193	1033796	6
5 3	1.60735	1033492	6
5 4	1.61833	1033328	6
5 5	1.6563	1033196	6
5 6	1.73695	1032712	7
6 0	8.87527	1248744	8
6 1	84.4128	1415064	8
6 2	75.6824	1345084	8
6 3	12.8119	1217348	8
6 4	16.0047	1204168	8
6 5	12.1129	1163732	8
6 6	4.20467	948164	8
6 7	3.5469	923880	8
6 8	2.16432	893844	8
6 9	1.79096	890288	8
6 10	1.61847	886752	9
7 0	940.722	3076400	10
7 1	980.753	3386972	10
7 2	949.03	3766328	10
7 3	765.999	4186912	10
7 4	919.21	3566128	10
7 5	909.391	2797636	10
7 6	949.651	2833228	10
7 7	915.495	3122060	10
7 8	904.28	2941244	10
7 9	921.426	3394876	10
7 10	903.815	2753488	10
7 11	902.267	2481868	10
7 12	27.2225	1039024	10

7 13	11.4055	1085616	10
7 14	2.75575	933540	11
7 15	1.85371	918040	11

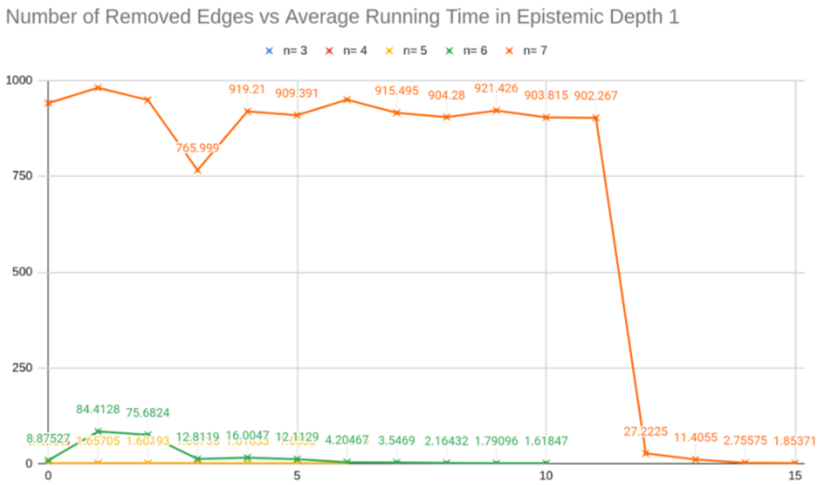
The results of the experiment in epistemic depth 1

APPENDIX C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3	1.41285	1.40112														
4	1.42248	1.48779	1.49373	1.49162												
5	1.6133	1.65705	1.60193	1.60735	1.61833	1.6563	1.73695									
6	8.87527	84.4128	75.6824	12.8119	16.0047	12.1129	4.20467	3.5469	2.16432	1.79096	1.61847					
7	940.722	980.753	949.03	765.999	919.21	909.391	949.651	915.495	904.28	921.426	903.815	902.267	27.2225	11.4055	2.75575	1.85371

Average running time (in seconds) of the experiment described above, at epistemic depth 1
X axis is the number of edges removed, Y axis is the number of agents

APPENDIX D



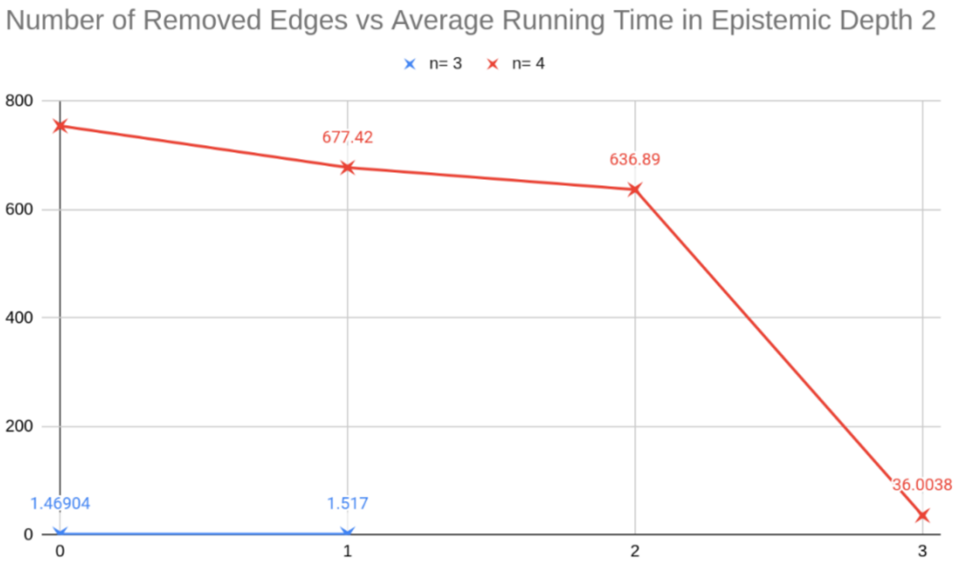
Result of the experiment at epistemic depth 1, visualized with a line chart

APPENDIX E

n, #removed edges	Average Total Time	Peak Memory	Plan Length
3 0	1.46904	803088	4
3 1	1.517	803708	4
4 0	754.152	4359380	6
4 1	677.42	4628392	6
4 2	636.89	4988432	7
4 3	36.0038	1579604	7
5 0	927.341	3228424	9
5 1	MEM	MEM	9
5 2	MEM	MEM	
5 3	MEM	MEM	
5 4	MEM	MEM	
5 5	MEM	MEM	
5 6	MEM, >3600s	MEM	

MEM: requires more than 32 GBs of memory
Results of the experiment at epistemic depth 2.

APPENDIX F



Result of the experiment at epistemic depth 2, visualized with a line chart

10. REFERENCES

1. Hajnal, A., Milner, E. C., &amp; Szemerédi, E. (1972). A Cure for the Telephone Disease. Canadian Mathematical Bulletin, 15(3), 447-450.
2. Picat. (n.d.). Retrieved November 08, 2020, from http://picat-lang.org/
3. Baker, B., &amp; Shostak, R. (1972). Gossips and Telephones. Discrete Mathematics, 2(3), 191-193.
4. https://ipc2018-classical.bitbucket.io
5. Cooper, M., Herzig, A., Maris, F., Perrotin, E., &amp; Vianey, J. (2020). Lightweight parallel multi-agent epistemic planning. Proceedings of the Seventeenth International Conference on Principles of Knowledge Representation and Reasoning.
6. Martin Cooper, Andreas Herzig, Faustine Maffre, Fr ́ed ́eric Maris, Pierre R ́egnier. A simple account of multiagent epistemic planning. 10`emes Journ ́ees Francophones sur la Planification, la D ́ecision et l'Apprentissage (JFPDA 2015), May 2015, Rennes, France. Actes des JFPDA 2015, pp. 23-29, 2015.
7. Raphael Yuster, Uri Zwick: Finding Even Cycles Even Faster. SIAM J. Discrete Math. 10(2): 209-222 (1997).
8. https://github.com/FaustineMaffre/GossipProblem-PDDL-generator