Initial Goal Allocation for Multi-agent Systems

Khanh Tra Nguyen Tran, Jonathan Young, Sravya Kondrakunta
St. Olaf College
1520 St Olaf Ave, Northfield, MN 55057

Abstract

In the multi-agent environment, a human expert engages in the allocation of objectives among individual agents. However, autonomous agents need to determine and allocate objectives without external intervention from humans. Therefore, in this research, we attempt to solve initial goal distribution challenges in multi-agent settings by developing two goal allocation algorithms. The primary objectives are to find cost-effective goal solution sets and distribute them evenly among available agents. We introduce two algorithms when the goals are structured in a hierarchical goal tree structure, and then test their efficiency across a variety of baseline allocation methods. Both algorithms were able to increase the performance of agents in multi-agent settings by finding the most optimal distributions of goals and allowing agents to act independently from human intervention.

Introduction

Goal management for autonomous agents in a multi-agent setting presents a challenge due to inherent uncertainties, remaining an open problem in the field of AI. One way to address this issue is to create a taxonomy of individual operations that play an important role in the decision-making of an agent (Cox and Veloso 1998). In the past, these goal operations have included goal formulation, goal selection, goal change, goal monitoring, goal delegation, and goal achievement (Cox, Dannenhauer, and Kondrakunta 2017; Kondrakunta 2017; Kondrakunta et al. 2021a; Aha 2018). Agents that possess these goal operations require less human intervention, enhancing their autonomy amidst uncertainties. In a similar context, groups of agents that can effectively distribute sub-goals in the pursuit of a larger goal would also increase the autonomy of the agents. The current set of goal operations is sufficient for multi-agent systems that already have their initial individual goals assigned by a human expert. However, this task becomes tedious for a human expert when such systems have large numbers of agents.

In a setting with, say 20 agents, there is no way to ensure that the initial goal allocation by a human expert is an error-free process. Even more so, we run the risk of completing sub-goals that contribute nothing toward achieving the main overarching goal. This is especially true when the goal structure of the environment is of hierarchical goal tree format (Shivashankar et al. 2013; 2014). For example, in surveillance tasks with high overhead communication costs-such as oceanic surveys, if you ask these 20 agents to scan 5 square miles of the ocean floor, it would make logical sense for each agent to only understand which square quarter mile it was responsible for scanning. In other words, each agent should be assigned distinct sub-goals that are necessary for completing the main goal to ensure minimal agent resources are wasted. If all 20 agents started in the same location and all had the same instructions, the locations around the starting position would likely be scanned multiple times while further quadrants may not be scanned at all. In this paper, we aim to solve the aforementioned issues by proposing a couple of initial goal allocation algorithms. We assume goals in the environment can be presented as a Hierarchical Goal Tree structure and agents have limited amounts of resources to achieve their assigned goals.

The paper is organized as follows: Section 2 introduces current goal operations and outlines the challenges in initial goal allocation in multi-agent settings. Section 3 introduces the Hierarchical Goal Tree format and defines the structure used in our research. Section 4 details the agent structure for goal distribution. Section 5 presents developed algorithms and two baseline algorithms. Section 6 covers efficiency test experiments and results. Section 7 discusses related work, leading to the conclusion in Section 8.

Goal Operations

Each autonomous agent in a multi-agent setting has access to a variety of individual operations during the decision-making process of an agent (Kondrakunta et al. 2021b). Goal operations are the actions that an agent performs in the pursuit of completing a goal. There are 6 goal operations in the goal reasoning research area (Aha 2018; Aha, Cox, and Muñoz-Avila 2013):

- **Goal formulation:** creation of new pending goals (Kondrakunta et al. 2021b; 2021a; Gogineni et al. 2018; 2019)
- **Goal selection:** choosing an active goal from pending goals. (Kondrakunta and Cox 2017; Johnson et al. 2016;

Gogineni et al. 2020)

- **Goal change:** changing the current goal to a similar one (Cox, Dannenhauer, and Kondrakunta 2017)
- **Goal monitoring:** watching that a current or pending goal is still relevant (Cox and Dannenhauer 2017; Gogineni et al. 2020)
- **Goal delegation:** distributing a goal to another agent (implemented in some certain domains)(Gogineni, Kondrakunta, and Cox 2021; Gogineni 2021)
- **Goal achievement:** executing actions to achieve a goal state (Cox 2017)

There has been prior research on goal operations as a whole and also individually on each goal operation. Up until now, it has been assumed that current goals are pre-allocated to agents in the beginning before other operations come into use. So, this paper focuses on the inherent uncertainty in the initial goal allocation problem in multi-agent systems before any progress is made toward completing them. Optimal goal allocation should minimize agent resource usage while ensuring an even distribution of goals among available agents to prevent idle agents.

Hierarchical Goal Tree and Confinements

We assume hierarchical goal tree formatting with limited agent resources. In this environment, all goals are organized into a hierarchical goal tree (HGT). Each goal in the HGT is interconnected, forming a tree structure where every goal is achievable by the agent. The main goal serves as the root, with sub-goals as its children, and primal tasks as its leaf nodes. Sub-goals represent incremental steps towards achieving the main goal. Thus, in the HGT, each goal has a parent-child relationship, with the completion of child goals contributing to the fulfillment of their parent goals.

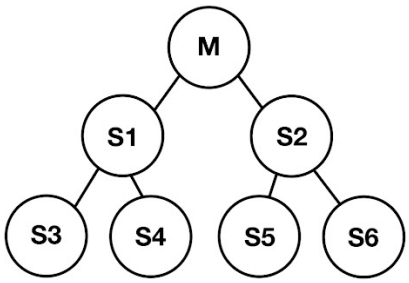

Figure 1: Example hierarchical goal tree where the main goal is the root and all sub-goals are smaller tasks. The leaf goals are considered 'primitive' tasks that cannot be broken down further.

Figure 1 depicts a simple binary hierarchical goal tree of depth three. Presented are the five ways to select which goals the agents need to accomplish to succeed in achieving the main goal: $M \parallel \{S1, S2\} \parallel \{S1, S5, S6\} \parallel \{S2, S3, S4\} \parallel \{S3, S4, S5, S6\}$

There are no scenarios where there are goals included in the solution that are not necessary. Each goal in a specific solution is required to complete the main goal successfully.

Each goal node object in the HGT possesses four attributes. The first attribute is an arbitrary name we assign to it to track which goals are processed. The second part is a collection of costs to achieve that goal for every available agent. Cost refers to the amount of resources an agent could spend to complete that task. This collection has a unique cost for each agent that's able to complete the goal. These costs are initialized at the creation of each goal and the costs can either vary from agent to agent or they can be equal across all agents. In other words, when the costs are not equal, there is a choice to be made to get the best-fit agent for each goal while when the costs are the same, all agents can achieve the goal for the same cost and any agent can be chosen regardless of the cost efficiency. The third part is a list of its child goal nodes if it has some sub-goals that we can achieve instead of the current goal itself. The last component is the value of discrepancy, which is the cost difference between the best-fit agent (spending the fewest resources) and the worst-fit agent (spending the most resources).

Agents and Confinements

Goals within the hierarchical goal tree are selected and distributed to the available agents for completion. Initial allocation assumes agents can fulfill their assigned goals, leading to the main goal's achievement upon their completion. We understand that in most situations, there is no way to understand all potential deviations from the original plan due to uncertain environments. However, at this point, we assume the plan will be completed without issue. We plan to introduce partial observability and the dynamic nature of the environment in the future.

Each agent within the domain is initialized with a name, the number of resources it possesses, and an empty list of goals assigned to it. Agent names are purely for convention and can be arbitrarily assigned without affecting the output. An agent's starting resources do indeed have an impact on the output. The number of resources allocated to an agent at any given time has a direct influence on the number of goals it can be assigned, or more specifically the cost assigned to the agent to be completed. Therefore, an agent that has more resources has more capability to complete more goals than an agent with fewer resources. When we refer to an agent's resources we are essentially speaking on the agent's battery or fuel tank. Having a larger battery or fuel tank allows an agent to complete more goals. The last part of each agent is the list of goals assigned to it. This list is initialized as empty since every agent is not assigned any goals in the beginning. When a solution is found for the hierarchical goal tree, the selected goals are then distributed to agents in the domain. Goals assigned to a specific agent are added to its list of goals. At no point during the initial allocation of goals will an agent be assigned more goals than it can complete regarding the agent's maximum number of resources. Goals are only assigned if the agent is capable and has enough resources to complete them.

Allocation Algorithms

We now delve into our developed algorithms. Our focus is to efficiently identify a unique set of goals within a hierarchical tree structure to achieve the main goal. These goals must be chosen in regard to keeping resource usage low while also evenly distributing them among agents. We have developed two unique algorithms that aim to address this problem.

Bottom-Up Allocation

As presented in Table 1 the Bottom-Up Allocation algorithm consists of two main cycles. The first cycle is concerned with selecting a set of goals that achieve the overarching goal and the second cycle finds a distribution of the goals among agents. When selecting goals to achieve, each goal is initially identified by the minimum completion cost of the available agents. Bottom-Up Allocation finds the absolute cheapest possible set of goals using a depth-first search by comparing each goal's cost against the sum of the goal's children's costs. If the goal itself can be achieved at a lower cost then it is returned, and vice-versa. This pattern continues upwards comparing the cost of child goals against the parent node until finally the cheapest set of sub-goals is compared to the cost of the root goal.

The second cycle begins by sorting the selected goals based on how much the goal's completion cost varies from agent to agent. The wider the variation, the more important it is to assign that particular goal to its best-fit agent. Once the goals are ordered the total cost of completing them is divided by the number of agents available in the domain. This is to help ensure that each agent is assigned an equal portion of the work. The next step is assigning the sorted goals to the available agent that can complete the goal for the cheapest resource cost. Agents are only available if assigning them that goal doesn't put them over their work percentage. If a goal can't be given to any agent due to work percentage it is moved into a leftover goals container. The final step of Bottom-Up Allocation is to assign the leftover goals to the current least-assigned agent. If this is not possible due to agents not having enough resources, the allocation fails.

In most hierarchical goal tree scenarios Bottom-Up Allocation provides an optimized solution in regards to both factors of keeping the total resource cost low while also evenly distributing the goals among agents. The best performances are found when each goal can be accomplished by a higher number of available agents and when agent resources are higher. In situations where this is not the case, Bottom-Up Allocation may not find the most optimal solution and may sometimes not find a solution when there is one. The tradeoffs of using Bottom-Up Allocation include having an even balance between total resource usage and even distribution of goals to agents for a much lower time complexity than a brute force approach, but sometimes sacrificing finding a solution when there is only a small number of solutions. Bottom-Up Allocation provides a middle solution between a completely greedy allocation and a completely even distribution. The upper bound runtime is $O(n(c+m))$ where n is the number of nodes, c is the number of child nodes, and m is the number of agents.

Table 1: Method for allocating goals to available agents. Parameters are the current goal tree and the available resources to each agent.

BottomUpAllocation (goal_tree, resources)

selected_goals = _optimal_path($goal\_tree$, $resources$)
  if not $curr\_goal.sub\_goals$ **then return** $curr\_goal$
  $selected\_goals$ = [ ]
  sub_goals = [ ]
  for sub **in** $curr\_goal$.sub_goals
    sub.append(_optimal_path(sub, $resources$)
  if $curr\_goal$.cost <= $resources$ **and** $curr\_goal$.cost < sub_goals.cost
    return $selected\_goals$.append($curr\_goal$)
  return $selected\_goals$.extend(sub_goals)
distributed_goals = _distribute_goals($selected\_goals$, $resources$)
  $assigned\_goals$ = {$agent$: [$goals$]}
  $goals$ = sorted($selected\_goals$.discrepancy)
  $sensitivity$ = sum($goals$.min_cost) / len($goals$)
  $left\_over\_goals$ = [ ]
  for $goal$ **in** $goals$
    if $assigned\_goal$ **then** $assigned\_goals$[$agent$].append($goal$)
    else $left\_over\_goals$.append($goal$)
  for $goal$ **in** $left\_over\_goals$
    if $assigned\_goal$ **then** $assigned\_goals$[$agent$].append($goal$)
    else return -1
  return $assigned\_goals$
return $distributed\_goals$

Resource-Conscious Allocation

The Resource-Conscious Allocation algorithm, depicted in Table 2, starts by iterating through every goal node in the Goal Tree and assigning the best agent, the agent that spends the fewest resources to achieve the goal, to accomplish that goal. Subsequently, the algorithm checks if the hierarchical goal tree is empty or not. If the goal tree is not empty, the function counts available agents and checks for disparities in their remaining resources. In the presence of a single available agent, the function performs goal allocation directly, skipping the equal goal distribution step.

In the cases with multiple agents, the function initializes a list of Goal Nodes with the root of the goal tree as the first element. Then, the function **_decision_algorithm** from Table 3 is called to allocate goals iteratively to agents. This also updates goal allocation for the current goal and remaining resources for each agent accordingly after each function call while handling unachievable goals. This process continues until no more goals can be allocated and the goals assigned to agents (or agents decided to achieve). After the allocation step is completed, it checks whether the final goal list is empty or not. If it is, the function returns nothing, which indicates that there is no goal node to be achieved and this goal tree cannot be accomplished. If the list is not empty, the function calls the function **allocate_goals_greedy** to improve the equality of goal distribution among agents, then repeating this process for all goals to distribute them as evenly as possible among agents while respecting resource

Table 2: Resource-Conscious Allocation. Parameters are the root of the goal tree and the list of the max resources of all available agents.

ResourceConscious(goal_tree, max_resources)

if $goal\_tree$ **is** NULL **do**
  raise Empty Tree Error Message
Let $i = 0$, $same\_resource$ = True, num_agents = len($max\_resources$)
while $i < (num\_agents - 1)$ **and** $same\_resource$ **is** True **do**
  $same\_resource$ **is** whether $max\_resources[i]$ and $max\_resources[i+1]$ are the same
  Increment i by 1
Let $agents$ = List of name of available agents
if num_agents **is** 1
  Let max_res = $\{agents[0]$: $max\_resources[0]\}$, $goal\_allocation$ = $\{agent[0]$: [ ]$\}$, $list\_goal$ = $[goal\_tree]$
  Let i = 0
  while size of $list\_goal \neq 0$ **and** $i <$ size of $list\_goal$ **do**
    $i, list\_goal, max\_res$ = _decision_algorithm($list\_goal, i, max\_res$)
    for $goal$ **in** $list\_goal$ **do**
      $goal\_allocation$[$goal$'s agent].append($goal$)
  return $goal\_allocation, max\_res$
Let max_res = $\{agent$: $max\_resources$[i] **for** i, $agent$ **in** enumerate($agents$)$\}$
Let $list\_goal$ = $[goal\_tree]$
Let $goal\_allocation$ = $\{agent$: [ ] **for** $agent$ **in** $agents\}$
Let $i = 0$
while size of $list\_goal \neq 0$ and $i <$ size of $list\_goal$ **do**
  $i, list\_goal, max\_res$ = _decision_algorithm($list\_goal, i, max\_res$)
 if $list\_goal$ **is** NULL **do**
  return () *// Return empty tuple*
$goal\_allocation$ = allocate_goals_greedy($list\_goal$, max_res)
return $goal\_allocation, max\_res$

constraints. There are five helper functions assisting with the main function of the algorithm.

The approximate runtime for this algorithm is $O(n \cdot a \log a)$ with n as the number of goals in the HGT and a as the number of agents. This runtime accounts for the agent-switching logic, the hierarchical tree traversal, and the greedy allocation process. This approach ensures that goals are assigned to agents only if they have sufficient resources to prevent resource exhaustion. This is also a dynamic decision-making process for goal allocation since it decides which goals to achieve and explores different combinations of agent-goal assignments considering the resource availability to optimize resource consumption. However, the algorithm's performance mostly depends on the initial conditions, such as the order or level of goals we sequentially consider in the list. Furthermore, the algorithm's complexity (space and time) is not optimized and gets increased proportionally with the hierarchical structure of the goal tree. The goal distribution among agents is not consistent in all goal distribution scenarios.

Table 3: Five helper functions assisting with the Resource-Conscious Allocation Algorithm

Function	Purpose
_check_resources	Verifying whether an agent possesses sufficient resources to attain a specified goal node
handle_unachievable_goal	Handling situations when a particular goal becomes unachievable (due to insufficient resources) by removing the unachievable goal from the list of goals and then subsequently assessing its children to identify achievable alternatives
calculate_tree_depth	Calculating the depth of a goal tree using recursion
_decision_algorithm	Determining the set of goals in the goal tree need to be accomplished to minimize the resources consumed and perform the allocation process on the goals
allocate_goals_greedy	Evenly distributing the goals determined from the _decision_algorithm among available agents regarding of the resources in a greedy manner

Random Allocation

We implemented a random goal-allocation strategy, which randomly assigned agents to specific goal nodes without consideration for resource optimization and equal goal distribution among agents.

The algorithm begins by determining whether the hierarchical goal tree is empty or not. Then it initializes an empty goal list and adds the root of the goal tree to the list. Then the code traverses the goal tree using a breadth-first search approach and assigning goals to the agent with the fewest resources, ensuring sufficient resources. The algorithm then utilizes the function **_decision_algorithm()** of the Recourse-Conscious Allocation Algorithm to traverse through the whole goal tree and select the goals that need to be performed for the most resource-optimized approach. If there is only one available agent, it assigns all the selected goals to the only agent, making sure that the agent possesses enough resources to achieve all the assigned goals. For multiple available agent settings, after having the list of optimized goals, we randomly pick an available agent to achieve each goal in the list of selected goals in the hierarchical order, ascertaining that the assigned agent has enough resources.

Since this algorithm incorporates randomness in assigning agents to goals, it benefits situations where goals are interchangeable and avoids predictable patterns in goal assign-

ments. However, the algorithm provides uneven goal distribution among agents, which could result in certain agents being overloaded with goals while others are underutilized. Also, though the goals are selected in the most optimized way in the beginning, the action of randomly swapping the agents to achieve each goal may not guarantee resource optimization.

Greedy Allocation

We implemented a greedy goal-allocation strategy which prioritizes optimizing the total resources the assigned agents consume to achieve the main goal without consideration for equal goal distribution among agents.

Since Greedy Allocation only considers the total resource usage of an allocation it is relatively simple. It begins similarly to Bottom-Up Allocation by using a depth-first search algorithm to find the cheapest possible solution set of goals of a hierarchical goal tree. Once the solution set of goals is selected they are distributed one by one in no particular order to their best fit agent. Greedy Allocation only considers whether an agent has enough resources to complete a goal before assigning it. If an agent does not have enough resources then the next best fit agent is selected. At any point, if a goal cannot be assigned because no agent has enough resources to accomplish it then the allocation fails.

Greedy Allocation thrives in scenarios where low resource consumption is the top priority. Greedy allocation finds the absolute lowest-cost solution to each goal tree but sacrifices any chance of evenly distributing the selected goals among agents. In some situations, Greedy Allocation may fail to find a solution when there is one if one agent is assigned the bulk of the work and as a result doesn't have enough resources for another goal that only that agent is capable of achieving. The major disadvantage of Greedy Allocation is the fact that in most allocations there will be many agents that aren't assigned any goals and as a result remain idle. This directly increases the amount of time a solution allocation would take to complete as fewer agents are working towards the achievement of the main goal.

Efficiency Tests

After completing both algorithms and the two baseline algorithms, we tested them against each other using a total of 8,000 different scenarios. Our tests consisted of three variables. The first variable is the number of available agents, the second involves uniform or varying completion costs for goals across agents (this allows for multi-agent scenarios when each agent has different capabilities), and the third pertains to whether all agents start with the same resources.

Test Configurations:

All of the tests we ran, each using 1000 unique trees.

- 3 agents, equal costs, same resources
- 3 agents, equal costs, different resources
- 3 agents, varying costs, same resources
- 3 agents, varying costs, different resources
- 1-10 agents, equal costs, same resources
- 1-10 agents, equal costs, different resources
- 1-10 agents, varying costs, same resources
- 1-10 agents, varying costs, different resources

Evaluation Metrics:

The metrics used in testing to measure the efficiency of each algorithm.

1. **Total Cost:** The total cost of completing all the selected goal nodes by the agent they're assigned to. (Lower is better)
2. **Agents Used:** The number of agents assigned a goal node to complete. (Higher is better)
3. **Discrepancy:** The difference in total cost between the agent with the largest workload and the agent with the smallest. (Lower is better)
4. **Cost Efficiency:** The difference between the total cost and the cheapest possible total cost without prioritizing even distribution. (Lower is better)

Empirical Results:

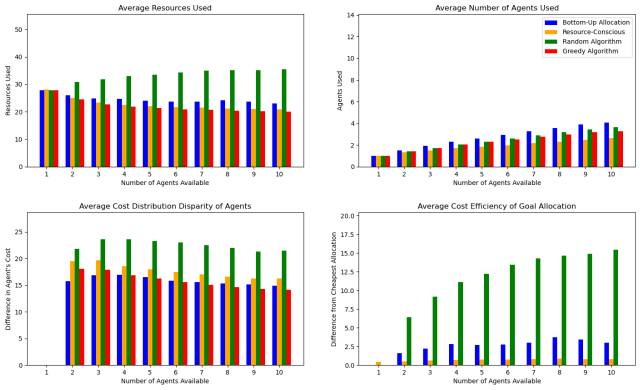

Figure 2: Agents With Same Max Resources Assigned To Goals (Different Resource Cost)

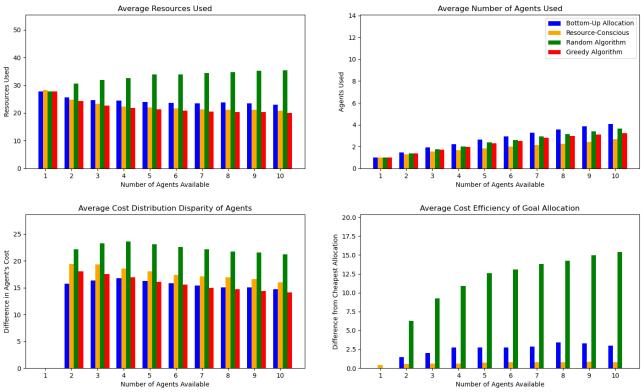

Figure 3: Agents With Different Max Resources Assigned To Goals (Different Resource Cost)

Description

We observed the same trend in these two scenarios above.

- Resource Costs: In most cases, the Allocation Algorithm exhibits the highest resource consumption while the Greedy Allocation Algorithm consumes the least resources. Regarding the two implemented algorithms, Resource-Conscious exhibits smaller resource consumption compared to the Bottom-Up Allocation Algorithm in most test cases, except the one-agent case. The Random Algorithm has the resource cost value directly proportional to the number of available agents. For the other three algorithms, the higher the number of available agents to accomplish the tree goal, the lower the amount of resources they consume to achieve all assigned goals.
- Agent Used: The Bottom-Up Algorithm consistently demonstrates the most equitable goal distribution among agents in two scenarios, even better than the Random Allocation. Conversely, the Resource-Conscious Algorithm exhibits the least efficiency in distributing the goals equally with the average number of assigned agents increasing slightly with the increase of available agents. Additionally, there exists a positive correlation between the number of available agents and the average number of assigned agents in all algorithms, but the rates of increase are diverse.
- Discrepancy: The Random Algorithm has the highest average of resource cost distribution disparity among the four algorithms, followed by Resource-Conscious. In contrast, the Bottom-Up and Greedy Algorithms have the least discrepancy, which shows these algorithms' efficiency in distributing goals to agents equitably. Regarding the trend of the average disparity of all four algorithms, it is directly proportional to the number of available agents when the number of agents is from 1 to 3 but after that, it gradually decreases with the increase of available agents.
- Cost Efficiency: The Greedy Algorithm exhibits no drift from the cheapest solution in all test cases due to its prioritization of the minimal total cost in goal distribution. This value can also be interpreted as the difference in resource costs between each of the three algorithms (2 implemented and Random algorithms) and the Greedy Algorithm pairwise. The Resource-Conscious Algorithm displays a unique behavior by being the only one having a drift in the test case with a single available agent while demonstrating the lowest drift in all subsequent cases. In conclusion, the observed trend in this graph mirrors that of the initial graph, which is caused by the correlation between cost efficiency and total cost.

Related Work

The concept of goal operations was first introduced by Cox and Veloso in 1998, where they discussed various goal operations and their significance. Although this idea has been present since then, recent research in the field of goal reasoning (Aha 2018) has gained significant traction. One extensively studied aspect of goal operations, whether explicit or implicit, is the problem of goal selection. Schank and Abelson addressed this by introducing goal types, prioritizing certain types over others. Another approach by Johnson et al. involved a cost-to-benefit analysis. Both of these approaches are tailored for single-agent scenarios and do not address the challenges of multi-agent systems or initial goal allocation. There is other research on goal change (Cox, Dannenhauer, and Kondrakunta 2017) and goal formulation (Kondrakunta et al. 2021b; Cox 2020) operations; however, they do not discuss the initial goal allocation problem. Dannenhauer, Molineaux, and Cox introduced the idea of goal monitors for goal achievement; once again, this research also does not focus on the initial goal allocation problem.

In the realm of multi-agent systems, our research aligns with the decentralized approach (Szymak 2012; Wooldridge and Jennings 1999), considering agents as independent, cooperative, or competitive based on the situation. While examining multi-agent goal management, there is limited explicit use of goal operations. Notably, Gogineni, Kondrakunta, and Cox explores goal delegation and sharing in a multi-agent scenario, assuming human expert distribution of initial goals. In contrast, our research focuses on solving the initial goal allocation problem within a multi-agent system.

Conclusion and Future Research

As AI becomes more prevalent in the world it's becoming increasingly important to find automated solutions to common problems. In this paper, we sought to address the problem of initial goal allocation and how goals should be distributed to a varying number of available agents in a domain.

Here we argued that our two implemented algorithms Bottom-Up Allocations and Resource-Conscious Allocation have made a positive impact on resource consumption, the number of available agents being used, and keeping agents workloads uniform. Both algorithms consistently perform better than the baseline algorithms when accounting for both low resource usage and even distribution among agents. Bottom-Up Allocation shows a bias toward achieving an even distribution by prioritizing involving another agent even if it slightly increases the total cost of the allocation. On the other hand, Resource-Conscious Allocation can achieve a lower total cost more consistently but may leave more agents idle. Therefore, these developed algorithms will benefit multi-agent systems that require initial goal allocation because agents will be better positioned within the domain and given tasks tailored toward their specific capabilities.

In the future, we plan to extend our research to include other goal operations. Additionally, our current assumptions include a perfect world; however, we plan to introduce the concepts of partial observability and dynamic environments to evaluate efficiency and enhance the algorithms further.

References

Aha, D. W.; Cox, M. T.; and Muñoz-Avila, H. 2013. Goal reasoning: Papers from the acs workshop (technical report cs-tr-5029). Technical report, University of Maryland, Department of Computer Science, College Park, MD.

Aha, D. W. 2018. Goal reasoning: Foundations, emerging applications, and prospects. *AI Magazine* 39(2)(2):3–24.

Cox, M. T., and Dannenhauer, Z. A. 2017. Perceptual goal monitors for cognitive agents in changing environments. In *Proceedings of the Fifth Annual Conference on Advances in Cognitive Systems, Poster Collection*. Palo Alto, CA: Cognitive Systems Foundation.

Cox, M. T., and Veloso, M. M. 1998. Goal transformations in continuous planning. In *Proceedings of the 1998 AAAI fall symposium on distributed continual planning*, 23–30.

Cox, M. T.; Dannenhauer, D.; and Kondrakunta, S. 2017. Goal operations for cognitive systems. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, volume 31, 4385–4391. Menlo Park, CA: AAAI Press.

Cox, M. T. 2017. A model of planning, action, and interpretation with goal reasoning. *Advances in Cognitive Systems* 5:57–76.

Cox, M. T. 2020. The problem with problems. In *Proceedings of the Eighth Annual Conference on Advances in Cognitive Systems*. Palo Alto, CA: Cognitive Systems Foundation.

Dannenhauer, Z.; Molineaux, M.; and Cox, M. T. 2019. Explanation-based goal monitors for autonomous agents. In *Advances in Cognitive Systems*, 1–6. Cognitive Systems Foundation.

Gogineni, V. R.; Kondrakunta, S.; Molineaux, M.; and Cox, M. T. 2018. Application of case-based explanations to formulate goals in an unpredictable mine clearance domain. In *26th International Conference on Case-Based Reasoning: Workshop Proceedings-Case-based Reasoning for the Explanation of Intelligent Systems*, 42–51. ICCBR.

Gogineni, V. R.; Kondrakunta, S.; Brown, D.; Molineaux, M.; and Cox, M. T. 2019. Probabilistic selection of case-based explanations in an underwater mine clearance domain. In *Case-Based Reasoning Research and Development: 27th International Conference, ICCBR 2019, Otzenhausen, Germany, September 8–12, 2019, Proceedings 27*, 110–124. Springer.

Gogineni, V. R.; Kondrakunta, S.; Molineaux, M.; and Cox, M. T. 2020. Case-based explanations and goal specific resource estimations. In *The Thirty-Third International Flairs Conference*, 407–412. AAAI Press.

Gogineni, V. R.; Kondrakunta, S.; and Cox, M. T. 2021. Multi-agent goal delegation. In *Proceedings of the 9th Goal Reasoning Workshop*.

Gogineni, V. R. 2021. *Goal Management in Multi-agent Systems*. Ph.D. Dissertation, Department of Computer Science & Engineering, Wright State University, Dayton, OH.

Johnson, B.; Roberts, M.; Apker, T.; and Aha, D. W. 2016. Goal reasoning with informative expectations. In *Proceedings of the 4th Annual Conference on Advances in Cognitive Systems*, 12: 1–14. Evanston, IL: Cognitive Systems Foundation.

Kondrakunta, S., and Cox, M. T. 2017. Autonomous goal selection operations for agent-based architectures. In *Working Notes of the 2017 IJCAI Goal Reasoning Workshop*. Melbourne, Australia: University of Maryland, Department of Computer Science.

Kondrakunta, S.; Gogineni, V. R.; Cox, M. T.; Coleman, D.; Tan, X.; Lin, T.; Hou, M.; Zhang, F.; McQuarrie, F.; and Edwards, C. R. 2021a. The rational selection of goal operations and the integration of search strategies with goal-driven autonomy. In *Proceedings of the Ninth Annual Conference on Advances in Cognitive Systems*. Cognitive Systems Foundation.

Kondrakunta, S.; Gogineni, V. R.; Molineaux, M.; and Cox, M. T. 2021b. Toward problem recognition, explanation and goal formulation. In *Proceedings from Advances in Artificial Intelligence and Applied Cognitive Computing*, In – Press. Las Vegas, NV: Springer.

Kondrakunta, S. 2017. Implementation and evaluation of goal selection in a cognitive architecture. Master's thesis, Wright State University.

Schank, R. C., and Abelson, R. P. 1977. *Scripts, plans, goals, and understanding: An inquiry into human knowledge structures*. Mahwah, NJ: Lawrence Erlbaum Associates.

Shivashankar, V.; Alford, R.; Kuter, U.; and Nau, D. S. 2013. Hierarchical goal networks and goal-driven autonomy: Going where ai planning meets goal reasoning. In *Goal Reasoning: Papers from the ACS Workshop*, 95.

Shivashankar, V.; Kaipa, K. N.; Nau, D. S.; and Gupta, S. K. 2014. Towards integrating hierarchical goal networks and motion planners to support planning for human-robot teams. In *AAAI Fall Symposium Series*, volume 4, 139 – 141. Arlington, VA: AAAI Press.

Szymak, P. 2012. Comparison of centralized, dispersed and hybrid multiagent control systems of underwater vehicles team. In *Solid State Phenomena*, volume 180, 114–121. Trans Tech Publications.

Wooldridge, M. J., and Jennings, N. R. 1999. The cooperative problem-solving process. *Journal of Logic and Computation* 9(4):563–592.