Evaluation and Adaptation of a Constraint Optimization and Distributed, Anytime A* Algorithm to Design-to-Criteria Scheduling Problem

by

Muhammad Asif Shiraz

A thesis submitted to the Graduate Council of
Texas State University in partial fulfillment
of the requirements for the degree of
Master of Science
with a Major in Software Engineering
May 2016

Committee Members:

Rodion Podorozhny, Chair

Guowei Yang

Mina Guirguis

FAIR USE AND AUTHOR'S PERMISSION STATEMENT

Fair Use

Duplication Permission

DEDICATION

Dedicated to my Teachers and Professors, especially Dr. Rodion Podorozhny

ACKNOWLEDGEMENTS

Acknowledgements are due to all the scientists whose work was referenced in preparation for this thesis.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Figure **Page**

LIST OF ABBREVIATIONS

Abbreviation **Description**

TAEMS - Task Analysis, Environmental Modeling and Simulation

FSM - Finite State Machine

DTC - Design-To-Criteria

APO - Asynchronous Partial Overlay

DCOP - Distributed Constraint Optimization Problem

SPAM - Scalable Periodic Anytime Mediation

QAF - Quality Accumulation Function

MQTT - Message Queuing Telemetry Transport

SAT - Propositional Satisfiability Problem

PB - Pseudo Boolean

ABSTRACT

Scheduling complex problem solving tasks where tasks are interrelated and there are multiple different ways to go about achieving a particular task is a computationally challenging problem. In this thesis, we study current approaches to solving such complex scheduling problems, and propose two new optimization techniques, which exploit A* based optimization, and constraint based optimization. We then perform an analytical comparison and computational complexity estimate for the efficiency enhancement achieved by these approaches, as compared against a base line case of "god's view" based optimal policy evaluation for same problems.

1. INTRODUCTION

Scheduling complex problem solving tasks where tasks are interrelated and there are multiple different ways to go about achieving a particular task is an imprecise science and the justification for this lies soundly in the combinatorics of the scheduling problem. Intractable problems require approximate solutions. Tom Wagner and Alan Garvey have developed a domain-independent approach to task scheduling called Design-to-Criteria (DTC) that controls the combinatorics via a satisficing methodology and custom designs schedules to meet a particular client's goal criteria. In their original approach to solve the Design-to-Criteria scheduling problem, criteria directed focusing approximation and heuristics in conjunction with soft goal criteria are used to make the scheduling problem tractable. Due to recent advances in efficient algorithms for solving distributed constraint optimization problems it might be much more efficient to solve the DTC scheduling problem using an approach that employs a distributed constraint optimization algorithm. An example of such an algorithm might be a Simplex method with some distributed negotiation algorithm such as Max-Sum, Adopt or APO. Mapping the DTC scheduling to a distributed constraint optimization algorithmic approach is non-trivial. It is the focus of the suggested proposal to create and evaluate such a mapping. Enhancing DTC scheduling using multiple optimization techniques based on constraint optimization is the focus of this study.

The Design-To-Criteria scheduling problem

This scheduling problem implies that execution constraints are defined in a goal tree in which each activity is described with domain independent attributes: duration of

the activity, quality of the activity's result and cost of the activity. The goodness of a schedule is defined via a utility function that calculates a quantitative evaluation of a schedule based on these three attributes of the activities in the schedule. The utility functions are domain-dependent, they are designed by a developer to express which schedule is more preferable, to introduce a way to compare schedules. The methods for choosing utility functions is not the focus of this work though. The nature of the problem domain dictates the nature of the utility function and, hence, the computational complexity of the algorithm that would deliver an optimal schedule. In practice, algorithms that would deliver an optimal schedule are intractable. This is due to the combinatorics mentioned above and this is also due to the fact that scheduling has a real-time constraint – it has to be done at certain frequency for iterations of the main control loop in case it is used for control of a robot or a team of robots.

The problem is called design-to-criteria because the utility function essentially adjusts the importance of the three attributes – criteria – in their contribution to the function's value for a particular schedule.

For this kind of problem, the computational complexity is dependent on the number of tasks and methods or subtasks present. If there are m subtasks of a parent task, and the sum QAF is employed, then the upper bound on all possible schedules is

$$\sum_{i=0}^{m} \binom{m}{i} i!$$

This is obviously an intractable problem for a brute-force methodology.

As mentioned, the original algorithmic approach by Wagner and Garvey to solving a DTC problem implied a state space search. First, the state space search

algorithm would generate alternative sets of activities recursively from the leaves to the root of a goal tree. At each node of a goal tree it would use a criteria-directed-focusing heuristic to avoid generating and propagating all possible alternatives. Next, the top alternative by the criteria-directed-focusing heuristic is used to build a schedule. The end-to-end schedule is built from the unordered activities contained in the alternative. Heuristic decision making is used to cope with the combinatorics and reason about the execution constraints expressed in the goal tree. This is the gist of the state space search based algorithmic approach.

Instead, this proposal suggests to map the goal tree and the utility function to a set of constraints that can serve as input to a constraint optimization solver. This mapping is the main contribution of this work. It will be evaluated in a realistic problem domain, for instance, distributed scheduling of robot activities to reach a combined goal.

Instead, this study suggests to augment the search of a mapped goal tree and utility functions with additional constraint optimization techniques using standard solvers like Sat4J to allow a mediating agent to resolve and minimize the impact of conflicts in a distributed schedule. Thus, our resulting algorithm combines the benefits of State Space Pruning and Constraint Optimization and A* shortest path search to deliver a vertically optimized distributed scheduling solution for autonomous multi-agent scheduling problems. It addition, the study goes a step further from mere simulation to deliver an enterprise class Java implementation, which was evaluated in a realistic problem domain denoting tasks that real robotic agents may need to solve in emergency assistance and associated vertical domains.

Distributed Constraint Optimization

Interacting agents, with design-to-criteria based goal trees for knowledge representation, will have local current schedules with constraints between activities that belong to different agents. For instance a constraint that some activity of agent A must be finished before some activity of agent B can be started (hard constraint). Each such inter-agent constraint has a number of satisfying solutions (e.g. times when activity of agent A can be done such that it is still finished before the given activity of agent B). A utility function for the combined schedules is likely to give different values based on different solutions to inter-agent constraints. That is, a combined schedule utility function has the solutions to inter-agent constraints as parameters. It is the job of a negotiation mechanism for a set of agents to converge to those solutions of inter-agent constraints that would maximize the utility function, in an environment where each agent holds a partial goal tree of the over-all problem set. One of the more obvious algorithmic approaches is binary search in the space of schedules. It works well for two agents. In case of more than two agents the binary search approach needs to be chained, this is not a good solution considering the real-time constraints for scheduling. Recent work in the distributed AI community has focused on mediation based approaches. A single agent from some neighborhood of agents (e.g. close by Euclidian distance) is chosen to "mediate". It polls the agents for their inter-agent constraints (not the complete goal trees, i.e. it does not create combined schedule for a set of agents). Once the mediator gets the constraints from all the neighborhood agents it, it works towards formulating a solution which combines constraint optimization techniques to develop a conflict free or reduced conflict allocation of tasks. Additional work is done by the mediation algorithm to accommodate

the nature of the utility function. The mediation agent uses an abstract utility function, which in our study, we have implemented using a few simplifying assumptions explained in detail later, but which do not compromise the quality and functioning of the algorithm if replaced by more complex logic.

So far most of the work linearized the utility functions because the simplex method does linear programming optimization. Even linearized optimization problem is very costly in time while most realistic utility functions are not linear.

In this work we propose to map the DTC problem into a distributed optimization from the start. Originally, DTC problem was solved locally and some binary search in space of schedules was used to arrive at combined, distributed schedules. We propose a novel approach in which the problem is divided into two separate stages, one of which provides a linearized abstraction of the underlying complex problem. We show that in this stage, optimization techniques can be plugged into the solution, to arrive at an approximate, high-level abstracted allocation of tasks to agents. We demonstrate this stage with a problem mapping that can then use standard third party libraries for the Max Sum algorithm to return a suggested allocation. The way to map the DTC problem to distributed constraint optimization is non-trivial.

Once this allocation is obtained, the agents then used an additional internal optimization based on an heuristic based enhanced Anytime formulation of the Dijkstra Algorithm, to efficiently develop an internal local schedule. The main differentiator in this algorithm is that

1) Creative distribution of the problem into two abstractions, one optimized via a constraint optimization technique, and the other optimized via Anytime task

scheduling

2) A unique mediation protocol that is not communication and data intensive, but provides sufficient information to the mediating agent to efficiently extract the heuristics it needs for applying at the optimization stage. This ensures conflict reduction in tasks shared by different agents.
3) An efficient internal scheduling algorithm for an individual agent to choose the best path for its local schedule, in real time manner.

Evaluation

In this thesis, we show how various inter-agent constraints can be separated from the constraints that can be solved locally by an agent.

While we demonstrate a mediator based approach, the criteria for an agent to become a mediator is not critical to the functioning of the algorithm, and thus, deployment can be adapted for a peer-to-peer environment where mediation becomes a dynamically assigned role.

The evaluation of the approach is done based on a realistic domain for control of a team of robots. For instance, the realistic domains can be transport and escort problem or distributed sensor problem or distributed mapping problem. It will be done by calculating am optimal schedule off-line, using a global constraint optimization (goal trees of all participating agents are combined). For instance, we modeled a transport and escort problem, which is more challenging and completely covers with in the simpler sensor problems discussed in some of the reviewed literature. This algorithm will run for the amount of time unacceptable for real time control but it will provide an optimal schedule,

but will be ill-suited for deployment in real time environment. Next we use the created distributed scheduling algorithm for the same scheduling problem. We collect the data about goodness of the schedules and amount of time taken, and build graphs that show how close the approximating distributed scheduling is to the baseline optimal one. Thus we obtain the quantitative evaluation of the developed algorithms are part of the comparison and evaluation of this work's result.

2. LITERATURE REVIEW

Roger Mailler has done extensive research on the subject of cooperative, distributed problem solving, in his notable doctoral dissertation [1]. He rightly notes that most of the prior work on mediation based approaches to problem solving focused on resolving conflicts, rather than developing cooperative synergies. In contrast, he proposed three new approaches for distributed problem solving, (APO), its optimized version (OptAPO), and real-time tailored Scalable, Periodic, Anytime Mediation (SPAM) algorithm.

In using the concept of a mediator in his literature, Mailler distinguishes between Competitive, Layered, and Cooperative Mediators, and it is this later type of competitive mediation, to which both his and our work focuses on.

Out of these three approaches, the one most relevant for our study is the SPAM algorithm, because it abandons solution convergence and optimization for sake for efficiency using "Anytime" characteristics, and targeting for deployment in a real time environment. The two major techniques used for this purpose are time boxing of the time to let the algorithm converge, and then dividing up the problem into discrete stages, each bringing a certain degree of quality to the solution, even if suboptimal.

One way in which this algorithm differs, or falls short of "as is" deployment in our target scenario is that its scheduling function is periodic, assuming that all agents start a particular task at the same time, thus leveraging a certain order in the events. In our case, tasks are continuously coming in and are handled as and when they arrive, thus covering a more realistic scenario than the abstract sanitized case of order triangulation measurements dealt with in Mailler's Thesis.

Another significant manner in which the SPAM algorithm differs from our solution is the restriction that mediating agents can only resolve conflicts among their immediate neighbors. This ensures that the size of the problem does not keep expanding to become unwieldy and time and resource intensive. In our case, however, we show how the need for such a restriction is obviated, because conflict resolution in our case follows a different pattern. Our agents are continuously moving (drones e.g.), as opposed to static agents (sensors e.g.) in case of SPAM. Thus the notion of immediate neighbors does not hold, and we allow our mediator agent to mediate for any one of its subordinates.

Farinelli et. al. (Farinelli 2014) have considered the problem of having physically distributed and computationally constrained devices make coordinated decisions to increase the effectiveness of the overall network of such devices. Their solution proposed the use of MaxSum algorithm, and similarly relied on DCOP heuristic to tame the computational complexity. Their work focused on multi-agent coordination using Distributed Stochastic Algorithm and Max Sum based message passing algorithm, both of which have comparable performance. In a similar study, (Fave, et al. 2012), the same approach was employed for unmanned aerial vehicles for disaster management, which is similar to the autonomous agents we consider in our work. Additionally, in their work, they also develop a framework to identify the different choices that need to be made when applying the solution. Our work focuses on those situations which they identified in their framework as modelling decisions as tasks, (rather than actions of the agent). However, since this paper only models aerial vehicles in the simulation, we cannot compare performance of a higher number of agents with our work. A similar study is done by (Ramchurn 2010) where the simulation environment was changed to RoboCup

Rescue Disaster Simulation Platform.

Since we are employing a constraint solver, the algorithms their optimization algorithms were also relevant to our research, notably (Rayside, Estler and Jackson, The guided improvement algorithm for exact, multi-purpose many-objective combinatorial optimization 2009)

Tambe et. al. (Tambe, Nair and Yokoo 2005) have presented a model which synthesizes Distributed Partially Observable Markov Decision Problems (Distributed POMDPs) with .Distributed Constraint Optimization to capture real world uncertainty of agent environments with the benefits gained by the knowledge of local interactions.

We also looked at approaches which employ decentralization without needing any communication between the agents, as discussed by (Berman, Kumar and Halasz 2009).

In terms of the architecture of the individual agents envisioned to act in a coordinated fashion, (Muscettola and al. 1998) have suggested interesting architectural and design insights that are useful to consider for implementation.

Macarthur et. al. (Macarthur and al. 2011) have introduced a novel distributed algorithm for multi-agent task allocation problem, which allows better scaling of the problem by utilizing pruning and reducing of the search space.

Kim et. al. (Kim and Lesser 2013) have presented some optimization techniques for multi-agent distributed constraint optimization, by performing a more efficient inference on which subset of values to consider without loss of accuracy.

3. PROBLEM DESCRIPTION

The problem we try to address in our research is a specific formulation of the more general scheduling problem. We have a number of agents, all working together to perform different tasks, with each task being composed of multiple sub-Tasks and methods. All the methods have a utility, giving the parent task a total utility which is an aggregate of the utility of the subtasks and methods.

We allow these problems to be specified in a standard TAEMS format (Horling, 1999), which is a way of modeling problem solving activities of intelligent agents, independent of domain specific semantics.

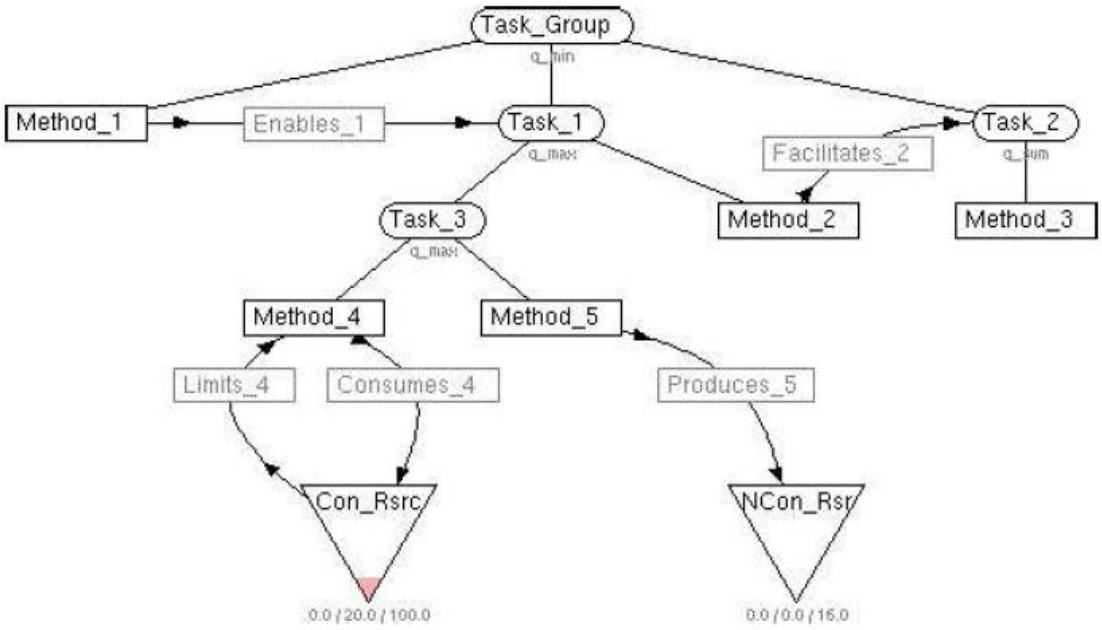


Figure 1 TAEMS structure example

In TAEMS environments, agents may be required to perform tasks which have specific deadlines, where information required for the optimal performance of a computational task may not be available and the results of multiple agent's computations

may need to be aggregated together in order to solve a high level goal. TAEMS representation is able to specify that there are multiple ways to accomplish a goal that trades off the time and resource cost for a certain result quality.

In TAEMS, the effect of an agent's activities may not be available from a local perspective. Instead, it must be measured from the perspective of how it contributes to the solution of a high level goal.

The goal of an agent is to maximize the quality achieved for each task group, before its deadline. A task group consists of a set of tasks related to one another by a subtask relationship that forms a graph. Tasks at the leaves of the tree represent executable amount of quality.

TAEMS Features

Tasks in a group may have different rules governing their execution and contribution to the goal quality, which are called Quality Accumulation Functions, or QAFs. Our solution supports three different QAFs, which are:

1) SumAll: Utility of all sub-tasks are added to determine the parent task quality. An example of such a task is a cooking recipe
2) SeqSum: Utility of all sub-tasks determines the parent task's quality, with the condition that all tasks must be executed in the sequence in which they are specified. Otherwise, no quality is accumulated.
3) ExactlyOne: Utility of the parent task is determined by the quality of any one of the sub tasks, and any one of the subtasks can be executed.

The implementation of additional QAFs is easily possible in the code base, since

the class architecture employs a generic interface that exposes the QAF's functionality.

Another important feature of TAEMS standard is the ability to define dependencies of tasks on each other. This can be at a task level, as well as at individual method level. Thus, one task must be completed before another can be started, otherwise the later will not accumulate any quality.

Yet another feature of TAEMS structures is ability to specify deadlines, which means that a certain task must be completed before a set duration. Thus, if there are a number of tasks to be scheduled, then those whose deadlines expire earlier, should be scheduled first, otherwise, after expiry, they will not accumulate any quality.

The goal in solving a TAEMS scheduling problem is to maximize the utility of all agents, which is measured by both the quality and quantity of the tasks which execute. The quality of a task is dependent on multiple factors, and is typically abstracted away from the primary scheduling algorithm, into a separate, pluggable, and extendable function which can be domain dependent and algorithm independent. Thus, the contribution of different factors into quality calculation can vary with different deployments of the algorithm to a real world problem. E.g. task of transporting a patient to a hospital will have the time factor playing a major role in its quality determination, but the task of filling up a gas tank will have gasoline price and distance from station as the main determinants of quality when comparing two possibilities. TAEMS supports the concept of duration, cost and quality, which constitute a tuple, whose values can be combined according to any custom rules defined in a utility function.

Sample Problem

Here is a sample problem, specified in TAEMS format. In real world, there would be a large number of tasks that need to be done, and more than one agent might be able to perform a task e.g. in the below example, in certain situations, a Police car may also be used to transport a patient to a hospital, while other tasks, like putting out a fire, are such that only a specific type of agent can do.

```
<Taems>
    <Task id="BaseTask" name="T3" qaf="SumAll" >
        <Task id="PickAndDrop" name="T1" qaf="SeqSum" >
            <Method id="M1" name="PickFromHouse" Quality="500" Duration="10" XCoord="250" YCoord="400"></Method>
            <Method id="M2" name="DropToHospital" Quality="500" Duration="10" XCoord="500" YCoord="350"></Method>
        </Task>
        <Task id="FillGas" name="T2" qaf="ExactlyOne" >
            <Method id="M3" name="FillFromStation1" Quality="500" Duration="10" XCoord="600" YCoord="250"></Method>
            <Method id="M4" name="FillFromStation2" Quality="500" Duration="10" XCoord="800" YCoord="300"></Method>
        </Task>
    </Task>
</Taems>
```

Figure 2 A Sample TAEMS task structure represented in XML format

In our solution, we employ a number of steps to calculate the best solution, using a distributed approach. Results of this distributed calculation is then aggregated and evaluated to find an optimum assignment.

In order to remain focused on the main problem at hand, we have made a number of simplifying assumptions which do not impact the functioning of the algorithms, but still allow the Java implementation to be remain manageable by reducing tangential features. Thus, our experiments and implementation employs the following conceptual elements, each of which conforms to their TAEMS based abstraction for scheduling problem solving. Above and beyond conformance to their TAEMS description, we further enhance the denotation of these entities in our analysis according to following descriptions.

1) Agents. Our agents are mobile, autonomous computing entities that implement the

IAgent interface, which defines some required functions which an agent must implement to be able to effectively participate in this problem solving approach described. However, actual details may vary depending on the real world vertical domain in which the agent is situated. Our Agent implementation veers towards mobile, autonomous agent similar to a vehicle or drone.

2) Communication. Agents can communicate with each other via standard, well defined messages, which require exchange of certain required data as part of the solution algorithm. The protocol can vary, and our tests were conducted over direct inter-thread communication, as well as MQTT (OASIS 2014) protocol based exchanges. MQTT is a light weight, open, and simple Client Server publish/subscribe messaging transport protocol, which is geared particularly for the emerging IoT (Internet of Things) applications. Our implementation can thus be deployed on multiple JVMs, each running an agent that can communicate over this protocol to collaborate for task scheduling and execution.

3) Tasks. Tasks are well defined activities which one or more agents must complete. Each task can be subdivided into multiple sub-tasks. And each Task has a utility value, which it contributes to the over-all schedule value if completed successfully. In our modelling, we have situated tasks for performance at particular locations, but this is not necessarily required in a generalized implementation of the algorithm. However, for such location free tasks, there must exist some other measure which determines when the cost of executing a task is higher than that of another one, which in our model is determined by the distance of its execution location from the Agent. Moreover, the tasks also have two more features supported by TAEMS, which are

quality (if that task is executed properly within it deadline) and a duration, which is the time it takes for an agent to perform it. The final utility value to be used in schedule computations is a combination of all of these factors, and the deployment domain determines how these relate to each other. E.g. tasks modelled in medical industry may attach high utility to minimizing duration, while industrial applications may attach higher utility to minimized resource consumption or cost.

4) Mediation. Since our algorithm falls in the category of a medication-based distributed problem solving technique, mediation is an important role which certain agents can perform. The elevation of an agent to this status, in our solution, can be owing to a number of factors, e.g. its location, its computational capacity or any other domain specific factor. All agents have the capability of becoming mediators, and so assignment of medication role to any particular agent as opposed to another does not impact the solution performance.

To illustrate the problem domain, we can create a very simple example. Assume there are two agents, A1 and A2 and three tasks need to be completed, T1, T2, T3. Task and Agent locations are shown in the figure below.

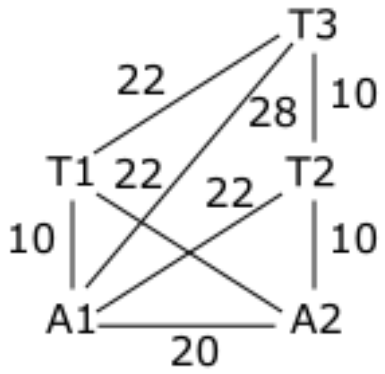


Figure 3 Example Task Execution scenario after injection of new tasks

In this diagram, there are two agents, A1, and A2, and there are different tasks that need to be performed. The time it takes for an Agent to reach the location for task execution is marked on the lines connecting the agent with the various tasks.

Utility of an agent performing a task is an abstraction for which any concrete domain dependent implementation can be plugged into the system. We follow a simplified example of making it directly proportional to task quality but inversely proportional to the time taken to reach its location and perform it. Thus, we define Utility in our implementation as:

$$U_{ta} = \begin{cases} 0, & t < L - T \\ Q_t - D_{ta}, & t \geq L - T \end{cases}$$

Where U_{ta} is the Utility of task t when performed by agent a, D_{ta} is the distance (or cost) of agent a performing task t, L is the deadline for the task and T is the current time.

If we set a uniform quality for all tasks to be equal to 50 units, then, intuitively we can tell that the best path in this scenario is for Agent A1 to take up T1, and Agent A2 to

take up T2 and T3. However, this is not always the case, if such simplifying assumptions are not made. Some of these complex scenarios in a real world application might be:

1. Task quality has an impact of which agent is performing it, e.g. a patient can be better transported in an ambulance, than a police car, as it can provide some interim care on the way to the hospital. Thus, when one agent calculates the quality, a higher value comes, offsetting a larger distance or cost.
2. There may be deadlines associated, so a task closer to an agent may wait, while it attends to a task much further away because its deadline is expiring.
3. TAEMS also supports interdependencies between tasks, which work pretty much like a deadline, so that a task's quality would be zero if performed before its prerequisite, but its full value if performed after it.

Algorithmically, however, the solution we have developed is independent of these vertical considerations which an agent may consider while calculating utility values. In real world, modern agent software will implement the IAgent interface which we have defined, and then some dependency injection based inversion of control mechanism might be employed to use our system for steering these agent's operations.

4. SOLUTION DESIGN

Our solution implementation starts with a dedicated task issuer, which is external to the system, and uses the MQTT protocol to receive new tasks into the system. This constitutes the boundary of the system, from where an external controller can feed tasks, either periodically, by human intervention (e.g. Console Input is also supported), or by invoking the interface based on events in another consuming system. In order to develop a fully functional system which can be deployed in the real world, we gave our task issuer the capability to directly assign a task to an agent, using the ASSIGN event type. This is because some tasks will have hard constraints that it can be performed by only a particular agent e.g. the example of a fire which can only be put out by a fire truck, and hence no negotiation is necessary if there is one fire truck type of agent. Obviously, if there are multiple fire trucks, then again, this task would not be directly assigned but entered for negotiation. Such direct assignment bypasses the system's capability of devising an optimal execution schedule itself, and so, for purposes of illustrating the functioning of our system, we will have the Task Issuer use the NEGOTIATE event type, when injecting new tasks. These custom events are broadcasted to all subscribers of MQTT, however, they are ignored by all agents, except the managing agent itself, who is responsible for conducting the negotiation.

Once these tasks arrive at the managing agent, it can process the tasks immediately, or queue them for periodic processing, if the rate of incoming tasks is too high. This can be set by an arbitrary configuration e.g. time to wait before initializing negotiation on tasks accumulated so far.

The managing agent has a list of all the agents which it is managing, and so it

issues a new message broadcast, requesting all agents to communicate to it what their incremental schedule qualities would be if they take up any combination of these new tasks.

Thus, if there are three agents being managed, an injection event by the Task Issuer results in three more messages by the managing agent, which are directed to each of the agents being managed. This brings us to the first portion of our scheduling solution, an A* based algorithm to calculate the best local schedule.

One additional enhancement possible in the mediation protocol is for an agent to query the mediating agent for new tasks if all of its current schedule is completed. In such a case, if no new tasks are available, then the mediating agent can poll all agents to "give up" one of their allocated tasks which are of lesser utility to them, or not of a critical nature. However, this step is useful from a practical standpoint, but not much from algorithmic research undertaken by us. So we did not implement this option. In our high traffic deployments, it may also not serve a useful purpose, as most agents are expected to remain busy.

A* Based Schedule Calculation

An agent's calculation of best local schedule is performed in two steps.

1) FSM Expansion. Transformation of TAEMS task representation into a Finite State Machine (FSM) of all valid execution paths respecting the given the TAEMS constraints.
2) A* application. In this step, we apply a customized Dijkstra based A* algorithm to this FSM, to find out the optimal path through this FSM, which will result in

the maximum schedule utility.

This algorithm is applied to the input tasks at two different times.

Once, when the managing agent requests cost calculation, and second, when an agent's cost calculation has "won" as task to be assigned to it, and so the same algorithm is applied during actual execution.

It remains a possible implementation optimization to cache results from a previous calculation. However, given the possibility that additional tasks may have been assigned between a cost feedback and eventual assignment, it is not necessary that the same schedule may necessarily have been valid. Thus, in our implementation, we reapply the algorithm in both stages separately.

Let's look at the two steps separately.

FSM Expansion

The TAEMS structure is a condensed representation of the tasks that need to be performed. It is possible for there to be multiple actual execution paths, all of which satisfy the constraints of a certain TAEMS structure.

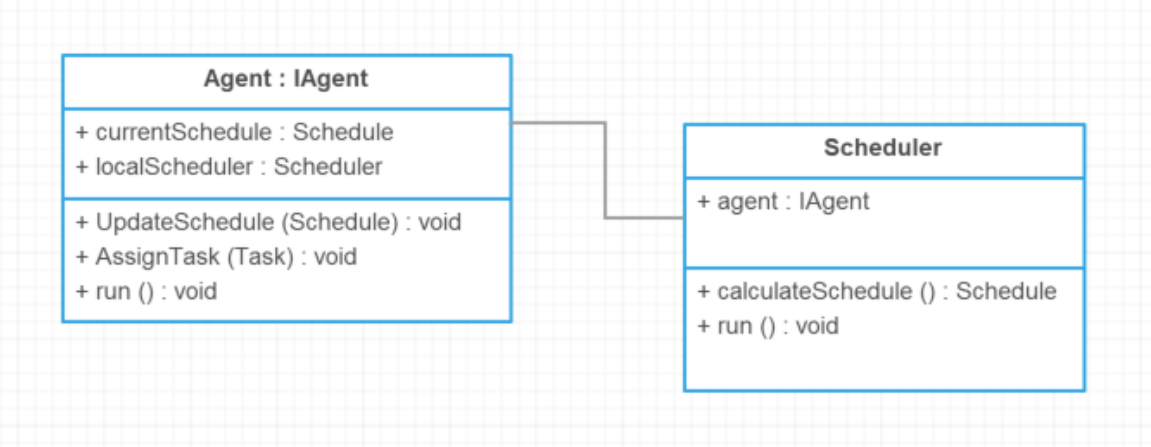


Figure 4 Agent and Scheduler simplified class diagram

In our implementation, an Agent object will have a separate scheduler, executing in parallel, so that the agent thread can continue executing the tasks, and a scheduler thread can continue calculating the best optimum schedule for the new threads that keep arriving, eventually to replace the reference in the Agent object, when it is in between two task executions.

Agent has a list of pending tasks, which are all Java entities representing the XML task structure of TAEMS. When the schedule is invoked to process these tasks, it performs the following operation, described in pseudo code:

```
Given: PendingTasks
Set CurrentTasks = Unexecuted Current Tasks
Set PendingTasks = Union PendingTasks and CurrentTasks
Set G = new Task Graph
For each (TaskXml in PendingTasks)
  Call ExpandTask(TaskXml)

Function ExpandTask(TaskXml)
  Set JavaTaskObject = Task Object Parsed from TaskXml
  Add JavaTaskObject to Graph G
  For each (ChildTaskXml in JavaTaskObject)
    Call ExpandTask(ChildTaskXml)
```

In our implementation, we have also used a third party library, GraphViz to visually illustrate the transformation of xml tasks into a Task Graph which the program expects. Such an FSM graph is illustrated below for the TAEMS structure in Figure 2.

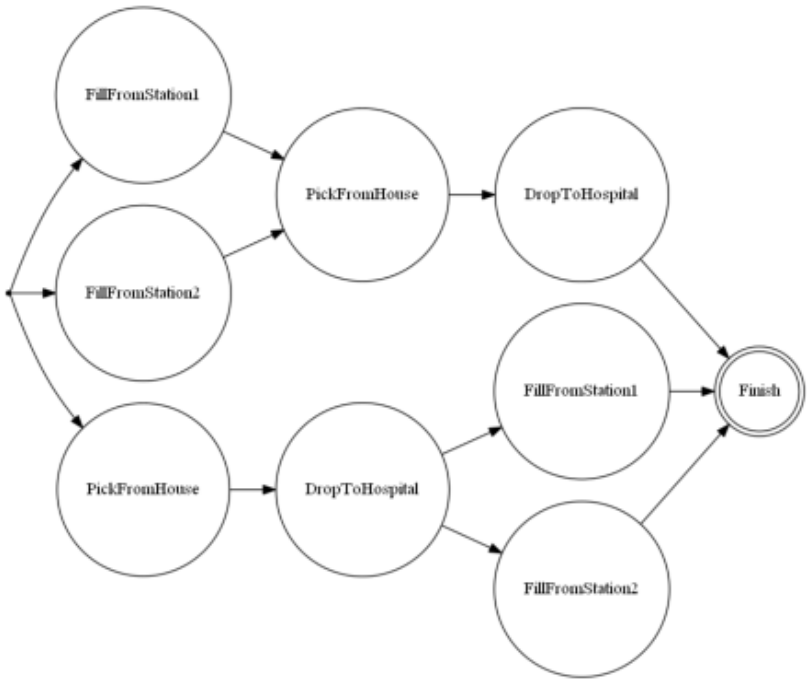


Figure 5 Finite State Machine of possible execution paths

As we can see from the task structure in figure 2, there is a base task with a QAF of SumAll, which means that all subtasks must be executed, and in any order. The main task has two sub tasks, to fill gas, and to drop a patient from his house to the hospital. The one two fill gas has a QAF of ExactlyOne, which means gas needs to be filled only once, and thus, either one of the two gas stations can be visited. The second task, is of type SeqSum, which means both must be executed in sequence: A patient must first be picked before he can be dropped.

Based on these QAFs, we see that the FSM expansion in Figure 6 respects all of these constraints. If we do the patient task first, and then we can finish by doing one more gas filling task. Or if we do one of the two gas filling tasks, then both of them lead on to dropping the patient task.

As may be obvious, as the number of tasks increase, the complexity of the FSM

keeps on increasing.

A* Calculation

Once the FSM has been expanded, then any path through that FSM is a valid execution path, and the only thing remains is to find the most optimal path through it. We use an adaption of the Dijkstra / A* based algorithm, which performs as follows:

```
Function A*(Method Source)
  settledNodes = new Set<Method>
  unSettledNodes = new Set<Method>
  predecessors = new Map<Method, Method>
  distance = new Map<Method, Cost>
  distance.put(source, Cost(source))
  unSettledNodes.add(source)
  while (Exist(unSettledNodes))
        Method node = getMaximumUtility(unSettledNodes, source)
     settledNodes.add(node);
     unSettledNodes.remove(node);
     findMaximumUtilities(node);

Function findMaximumUtilities(Method node)
  long accumulatedDuration = 0
  List<Method> adjacentNodes = getNeighbors(node)
  for (Method target in adjacentNodes)
     Cost highestUtilityToNode = getHighestUtility(node)
     Cost singleStepCost = getCost(node, target, highestUtilityToNode)
     Cost currentHighestUtility = getHighestUtility(target)
     Cost newUtility = singleStepCost
     If newUtility > currentHighestUtility Or IsFinalPoint(target)
          distance.put(target, newUtility)
```

PutPredecessor(target, node)
unSettledNodes.add(target)

Most of the features in this algorithm are standard A* steps, with a few customizations.

First off, the structure of the algorithm has been changed from calculating "minimum distance" between two points, to having "maximum utility" between two points on the graph. This is done by introducing a new construct for representing a "distance" for use in the algorithm, which is actually the utility based on any custom combination of the Task quality. It is an interface, whose concrete implementation can vary depending on the Agents and Tasks involved, making the algorithm generic.

Second, we introduce a conditional to check if the point is final point of the graph, where all paths are supposed to combine, and the best path will be measured from this end task, a hypothetical task representing completion of all others.

Lastly, we introduce a heuristic which again will be dependent on the vertical domain in which the solution is deployed. It denotes the expected utility of performing a certain action, with its value adjustable to balance optimality vs time performance, desired by the application. Our research focuses on distributed selection of best combination of Task assignments, and thus works on the proposed schedules which the agents return. How the agents themselves arrive at this number is a parallel concern to our primary focus. We have developed an efficient A* implementation, but it can be replaced by any other calculation including plain Dijkstra based solution, or the various other variations of A* proposed by (Likhachev 2005).

The output of this algorithm is a schedule, which contains all Methods to be executed and in the order in which they will be executed. This method chain will span all the tasks, and will be a linearized view of the path.

For any set of tasks received by an agent, it is unknown which subset of these tasks would be optimal for execution by this agent. Thus, the agent calculates a combination of all possible tasks, and performs the above calculation for each of them. And finally, it communicates back to the managing agent, a list of Qualities, along with the Task subset, which the managing agent then uses to decide an allocation. This quality chart is broadcast via an MQTT message.

Constraint Optimization Based Allocation

When the mediation agent has received feedback from all agents, regarding their respective schedule qualities for a task set, then it concludes the session by examining these results, and deciding which agents would be the best ones to be given which task. The negotiation thus ends with a hard assignment event, which results in the assigned agent executing the task.

This is done by mapping the results into a standardized constraint optimization problem, which is one of the primary contributions of this thesis. We propose a novel mechanism of using pseudo Boolean optimization technique to determine optimal assignment.

We researched different approaches to optimizing this step, and initially tried using Max Sum algorithm, using a third party library developed by Marc Pujol called jmaxsum (Pujol, 2015) (Pujol, 2015). However, as our research progressed, we further

improved our solution by using Sat4J Pseudo Boolean Satisfiability library (Parrain 2010).

Sat4J performs optimization by the strengthening procedure, where it calculates a satisfaction solution, and then introduces a constraint to prevent calculation of solutions greater than that. Once a solver cannot find any better solution, it returns the best solution found.

There are other distributed SAT approaches also (Ruiz 2011), however, the library we chose does not work distributedly.

In order to use the Sat4J solver to our problem, we first need to transform our problem as input to the solver, and then transform the solver output back into a solution to the problem.

A standard formalization of our problem input, which consists of the data collected by the managing agent, can be expressed as the set S such that:

$S = \{D_1, D_2 \dots D_a\}$ where

$A = \{\text{Set of all agents}\}$, $a = |A|$

$T = \{\text{Set of all tasks to be executed}\}$, $t = |T|$

$D_i = \{C_1, C_2, C_3 \dots C_n\}$ where $n = 2^t - 1$

$C_i = \{U, T\}$ where U is the utility when tasks T are executed by this agent, with

$T \in \binom{T}{k}$ and $k \in \{1, 2 \dots t\}$

Solver Mapping

Now with this input data set, we need to map it into an input for the PB solver. The Sat4J solver we use accepts a variant of the OPB format (Roussel 2007). This format

requires the input to be divided into three sections.

1) Comments: The first line has to be comments, which can contain any descriptive data, but has one required metadata portion which tells the solver how many variables we have in our problem, and how may constraints we are supplying. This allows the solver to function more optimally instead of having to reflect on this data from the problem. An example of this metadata comment is:

   * #variable= 5 #constraint= 4

   Comments always start with a * symbol.

2) Second section contains the objective statement, which we need to minimize. The objective statement must contain variables less than or equal to the number of variables we declared in the comments. E.g. following is an objective function

   min: +1 x1 +2 x2 +4 x3;

   Here the first signed number is the coefficient, whose variable is expressed after a space, and using a variable name that must start with an x with any number, usually kept sequential.

3) Last section consists of constraint statements. Each constraint appears on a single line, and express an inequality which must be satisfied by the solution. A constraint looks like this following:

   +1 x7 -1 x1 -1 x4 >= -1;

The output of the solver is of the form [1|0] x_1, [1|0] x_2 … [1|0] x_i where all variables are prefixed with a 0 or a 1, depending on which of these values allow the objective function to be minimized, and simultaneously satisfy the constraints.

In order to map our task scheduling problem to an input for the solver, we need to

transform it in such a way that the output can help us extract an optimal scheduling.

This implies that the variables should represent all possible agent and task allocations, and the objective function should be such that once a subset of these variables are selected for the function's minimization, that selection should correspond to the best possible schedule.

The solution we have devised uses one variable to represent the allocation for one individual agent. Thus, if there are two tasks to be assigned, an agent can perform either one of them, or both. Thus, we introduce three variables, each representing the allocation of tasks to one single agent. Our variables are thus the sets of different tasks that can be performed by a particular agent. The number of variables will be equal to number of agents times all possible combination of tasks which that agent can perform. The coefficient will be the sum of the utilities when the agent performs those particular tasks. Since the utilities are positive values, and the solver supports minimization, so we reverse the sign of the utilities values in the objective function, to make minimization simulate maximum utility.

The number of constraints in our mapping, will be equal to the number of tasks. One constraint will express the fact that one task can only be executed by one agent, and that multiple agents cannot be asked to perform the same task, otherwise, its quality will be accumulated twice. The variables participating in a constraint will be only those variables, whose corresponding Task sets contain the task being represented in this constraint.

For the example scenario discussed earlier, we can look at the table below to understand how this mapping works.

Table 1 – Agent and Task mapping to PB Format

Agent A1	A1 Utility	Mapping	Agent A2	A2 Utility	Mapping
T1	(50x1)-10=40	x1	T1,T2,T3	(50x3)-22-20-10=98	x8
T2	(50x1)-22=28	x2	T2,T3	(50x2)-10-10=80	x9
T3	(50x1)-28=22	x3	T1, T3	(50x2)-22-22=56	x10
T1, T2	(50x2)-10-20=70	x4	T1, T2	(50x2)-22-20=58	x11
T1, T3	(50x2)-10-22=68	x5	T3	(50x1)-20=30	x12
T2, T3	(50x2)-22-10=68	x6	T2	(50x1)-10=40	x13
T1,T2,T3	(50x3)-10-20-10=110	x7	T1	(50x1)-22=28	x14

* #variable= 14 #constraint= 3

* this is a sample PB input format

min: -40 x1 -28 x2 -22 x3 -70 x4 -68 x5 -68 x6 -110 x7 -98 x8 -80 x9 -56 x10 -58 x11 -30 x12 -40 x13 -28 x14;

1 x1 1 x4 1 x5 1 x7 1 x8 1 x10 1 x11 1 x14 = 1;

1 x2 1 x4 1 x6 1 x7 1 x8 1 x9 1 x11 1 x13 = 1;

1 x3 1 x5 1 x6 1 x7 1 x8 1 x9 1 x10 1 x12 = 1;

The data in this table is held in memory before the solver is invoked, and after the solver returns a preferred combination of variables, we map those variables back into the task allocations represented by them. The constraints ensure two things:

1) That each task is considered in the solution. This is because each constraint must have one variable selected from it, so that its Boolean value becomes 1. If a variable from each constraint is selected, then it ensures that each task is represented, because one constraint only contains variables that have one particular task commonly appearing

among them.

2) That each task is considered only once. Tasks are contained by variables, and the same variables can repeat across constraints also. E.g. x7 appears in all constraints in the example above. However, if x7 is selected in one constraint, and then it ensures no other variable is selected in any of the constraints, otherwise, that other variable's coefficient, and x7's coefficient will both combine to make that constraint become 2, instead of 1, and thus be a violation.

When this input is given to the solver, it comes back with the following solution and objective function value:

Solution: x1 –x2 –x3 –x4 –x5 –x6 –x7 –x8 x9 –x10 –x11 –x12 –x13 –x14
Objective function = -120

So the minimized value it has found is -120, which occurs when x1 and x9 are given a positive coefficient, and all others variables are given negative coefficients. We then translate this result back into our problem, by using the table mapping of variables to task allocations. We see that variable x1 indicates that T1 is assigned to A1 with a utility of 40, and variable x9 indicates that T2 and T3 are assigned to A2 with a utility of 80, making the total highest utility to be 120 units.

Once the managing agent has applied this technique to determine the bets schedule, it broadcasts an ASSIGN MQTT message, which concretely assigns these tasks to the selected agents. After receiving this hard assignment, the agents put these new tasks into their list of pending tasks. A separate scheduler thread running on the agent calculates a new schedule for the agent, taking into account these fresh tasks, and then merges this schedule back into the agent's current execution schedule, at the first

opportunity it gets (which is basically an interval between the agent finishing one task and going to next).

Execution Example

In this section, we will walk through two complete execution scenarios, and explain the intermediate results at each step. The first scenario is when the agents are free and new tasks are injected, and the second is where agents already executing on a schedule are required to update their current schedule based on availability of new pending tasks.

Events processed by the system are numbered to demonstrate the sequence.

1. Two agents A1 and A2 are initially located at positions [X150, Y150] and [X300, Y200]. A1 is acting as the managing agent.

2. The first wave of Task injection occurs, and four new tasks, T1, T2, T3, T4, are injected into the system, by sending the following message to agent A1, the managing agent.

   A1,NEGOTIATE,::::-T4-T1-T3-T2-

   <>

   This message is a custom format to indicate to agent A1 that it needs to negotiate execution of these four tasks among its sub-agents. Additional details can be appended into the message body, but in this case, it is empty. The Tasks themselves are referenced by name only, because it is assumed that in a vertical implementation domain, agents will use a custom format to express the specialized knowledge of needing to know how to perform task. Thus, pre-requisite data for task execution constitutes of two components: Taems specification for the task, and additional customized knowledge. We assume that agents will know how to perform a particular task based on its reference name, and use a file based repository of custom Taems

data to express this knowledge, available to all agents. In some real world implementations, this data can also be embedded in a more elaborate MQTT messaging format to be employed.

The initial set of injected tasks are shown in figure below:

Figure 6 Initial Task and Agent Locations

3. Agent A1 first calculates the quality expected to be accrued by continuing on its current schedule. Since this is the initialization, current schedule quality value is 0.
4. Agent then combines the pending tasks from the current schedule with those just injected, to obtain a new set of pending tasks. In this instance, it will be the same as the set of newly injected tasks.
5. In order to calculate the quality expected to be accrued by executing all combinations of these tasks, agent first calculates S = ℙ(T), the power set of all the tasks allocated to it. This set includes following combinations: {T2}, {T2, T4}, {T2, T1}, {T2, T1, T4}, {T2, T3}, {T2, T3, T4}, {T2, T3, T1}, {T2, T3, T1, T4}, {T3}, {T3, T4}, {T3, T1}, {T3, T1, T4}, {T1}, {T1, T4}, {T4}
6. For each element in S, Agent expands the Taems tasks into an FSM. Two of these are shown in Figure below. To do this, it first initializes a dummy starting point method, and then applies a recursive algorithm to the task set. It traverses down the task tree, and for each method, appends it to the dummy starting point. This process is done differently for QAFs. SeqSum tasks are appended in order, SumAll are appended in

parallel with a series for each permutation, and ExactlyOne are appended in parallel with one task in each series. If, instead of a method, it encounters a sub-task, then the recursion is applied again at that task level. Two examples of FSM expansion of a simple one-method based task set of our example is shown below.

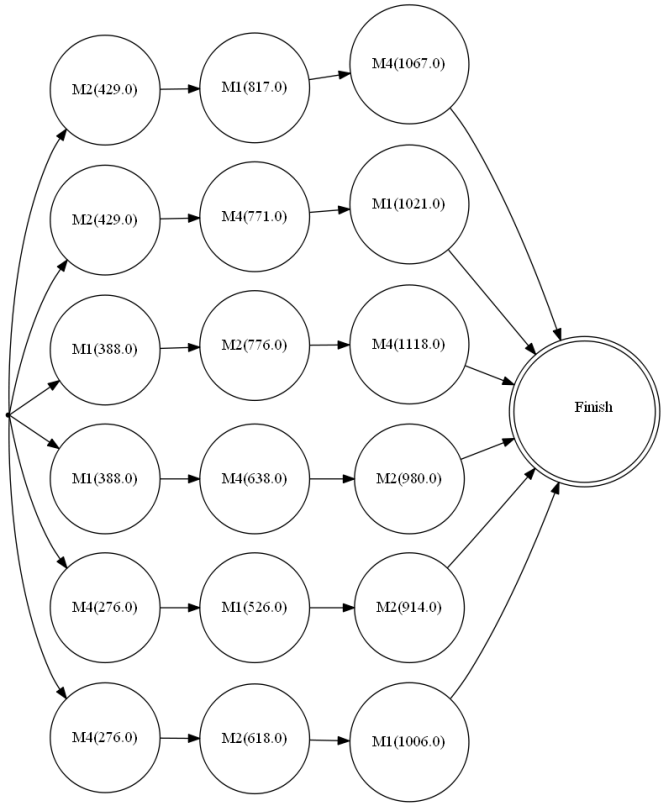


Figure 7 Expanded FSM for three tasks

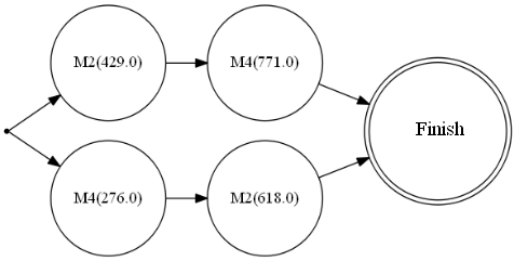


Figure 8 Expanded FSM for two tasks

For each of the FSMs, the agent applies the A* algorithm to find the best possible route, i.e. one accruing the highest quality. First we add the dummy starting point as a settled node, and then find the shortest path from it to the next node. The A* concept of shortest path is equivalent to the highest utility for our purpose. Thus, we define a custom class and its value is determined by a method supplied by the agent, which takes into account the cost, quality and time, as well as other parameter, including Taems concept of task deadline and an A* heuristic. If the heuristic, determined by factors peculiar to the deployment domain, determines that the quality accrued is less than the applied heuristic threshold, then the method returns with an outcome representing low quality, to discourage exploration of this path in future. In our case, we make distance proportional to cost and time, and quality proportional to a custom value we supply in the Taems XML. In the figure below, we see that the selected route, highlighted in green, has the highest accumulative quality of 1118 points.

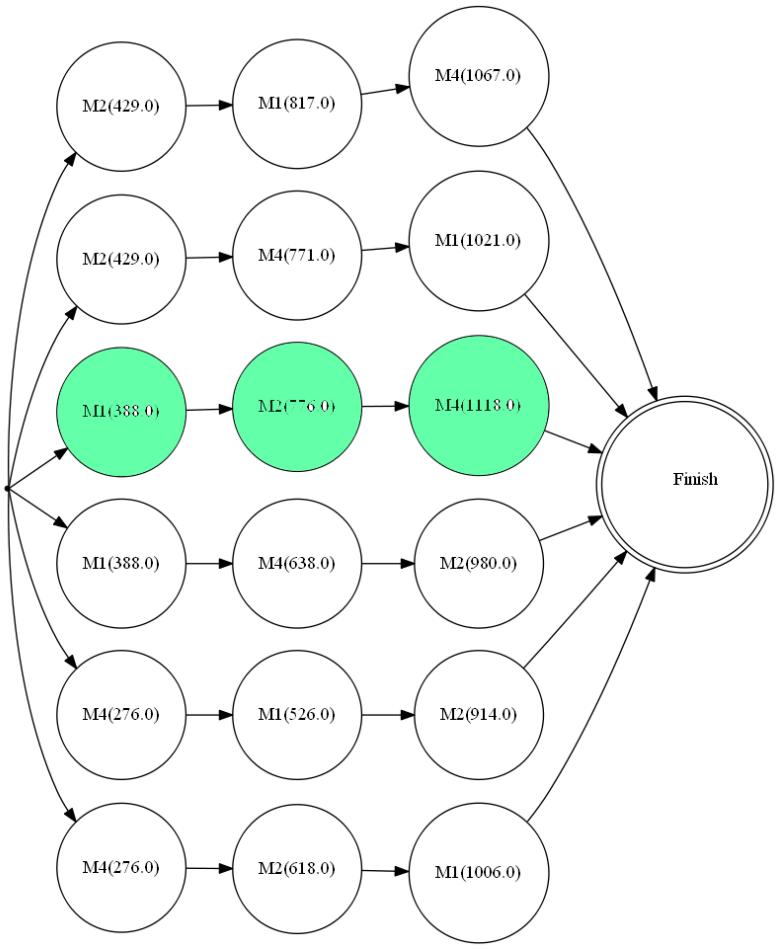


Figure 9 Expanded FSM for seventh step

7. The agent calculates schedule quality value of all the expanded FSMs, and then saves them in an internal data structure, whose values are as below. This represents an array of elements, each representing a structure indicating which Task Set, has how much base quality and how much incremental quality.
   [T2,0,429]
   [T2-T4,0,771]
   [T2-T1,0,817]
   [T2-T1-T4,0,1118]
   [T2-T3,0,780]
   [T2-T3-T4,0,1065]
   [T2-T3-T1,0,1127]
   [T2-T3-T1-T4,0,1412]
   [T3,0,365]
   [T3-T4,0,650]
   [T3-T1,0,464]
   [T3-T1-T4,0,931]
   [T1,0,338]
   [T1-T4,0,638]
   [T4,0,276]

8. After calculating qualities for itself, the agent then sends a message to all agents under its management to send back their quality values. This message looks as like this:
   A2,CALCULATECOST,::A2:::::A1
   <>
   which means that the message is directed to A2, who is required calculate costs and return them to A1.
9. A2 receives the message, and performs the same steps as described in Steps 4 to 7. Afterwards, unlike A1 (the managing agent) which saved the results itself, A1 sends

the results back to A1 in a directed MQTT message as follows:

A1,COSTBROADCAST,::A1::-T4-T1-T3-T2-:::A2

<[T2,0,359]

[T2-T4,0,701]

[T2-T1,0,747]

[T2-T1-T4,0,1072]

[T2-T3,0,787]

[T2-T3-T4,0,1129]

[T2-T3-T1,0,1175]

[T2-T3-T1-T4,0,1451]

[T3,0,436]

[T3-T4,0,721]

[T3-T1,0,694]

[T3-T1-T4,0,971]

[T1,0,250]

[T1-T4,0,595]

[T4,0,342]>

Note that the values calculated by A2 are different from those of A1 e.g. when A2 performs all tasks, it accrues a quality of 1451 points, as compared to A1's 1412.

10. Once A1 receives this message, it evaluates if calculations from all agents have been received. Since in our example, there is only a single agent being managed, the data collection is complete and A1 is ready to allocate the tasks.
11. A1 first creates a mapping table to assign a numbered variable e.g. x1, x2 etc. to each possible task allocation to an agent. The number of these variables for two agents and four tasks is 2 * ((4) + (5) + (4) + (1) + (1)) = 30. This corresponds to the product of number of agents multiplied by all possible allocations for it i.e. four possibilities of single task assignment, five possibilities for two tasks, four possibilities for three tasks, one possibility for four tasks, and one possibility for no task to be assigned to this agent. Thus we have variables from x1, x2 … x30

12. The agent then maps the problem to a SAT problem, where the objective function is to minimize the negative of the sum of all qualities accrued by the agents, considering all possible allocation combinations, but constraining it to have one task performed by one agent alone. The SAT problem in OPB format becomes as follows:

    * #variable= 30 #constraint= 6

    min: -429 x1 -771 x2 -817 x3 -1118 x4 -780 x5 -1065 x6 -1127 x7 -1412 x8 -365 x9 -650 x10 -646 x11 -931 x12 -388 x13 -638 x14 -276 x15 -359 x16 -701 x17 -747 x18 -1072 x19 -787 x20 -1129 x21 -1175 x22 -1451 x23 -436 x24 -721 x25 -694 x26 -971 x27 -250 x28 -592 x29 -342 x30;

    * x1=1[2] x2=1[2, 4] x3=1[2, 1] x4=1[2, 1, 4] x5=1[2, 3] x6=1[2, 3, 4] x7=1[2, 3, 1] x8=1[2, 3, 1, 4] x9=1[3] x10=1[3, 4] x11=1[3, 1] x12=1[3, 1, 4] x13=1[1] x14=1[1, 4] x15=1[4] x16=2[2] x17=2[2, 4] x18=2[2, 1] x19=2[2, 1, 4] x20=2[2, 3] x21=2[2, 3, 4] x22=2[2, 3, 1] x23=2[2, 3, 1, 4] x24=2[3] x25=2[3, 4] x26=2[3, 1] x27=2[3, 1, 4] x28=2[1] x29=2[1, 4] x30=2[4]

    1 x3 1 x4 1 x7 1 x8 1 x11 1 x12 1 x13 1 x14 1 x18 1 x19 1 x22 1 x23 1 x26 1 x27 1 x28 1 x29 = 1;

    1 x1 1 x2 1 x3 1 x4 1 x5 1 x6 1 x7 1 x8 1 x16 1 x17 1 x18 1 x19 1 x20 1 x21 1 x22 1 x23 = 1;

    1 x5 1 x6 1 x7 1 x8 1 x9 1 x10 1 x11 1 x12 1 x20 1 x21 1 x22 1 x23 1 x24 1 x25 1 x26 1 x27 = 1;

    1 x2 1 x4 1 x6 1 x8 1 x10 1 x12 1 x14 1 x15 1 x17 1 x19 1 x21 1 x23 1 x25 1 x27 1 x29 1 x30 = 1;

1 x1 1 x2 1 x3 1 x4 1 x5 1 x6 1 x7 1 x8 1 x9 1 x10 1 x11 1 x12 1 x13 1 x14 1 x15 = 1;

1 x16 1 x17 1 x18 1 x19 1 x20 1 x21 1 x22 1 x23 1 x24 1 x25 1 x26 1 x27 1 x28 1 x29 1 x30 = 1;

13. The solver returns the following result:
    [-1, -2, -3, 4, -5, -6, -7, -8, -9, -10, -11, -12, -13, -14, -15, -16, -17, -18, -19, -20, -21, -22, -23, 24, -25, -26, -27, -28, -29, -30]

14. This indicates that the best allocations are represented by the positive variables x4 and x24. This gives the allocation that A1 should be assigned T1, T2, T4 and A2 should be assigned T3.

15. A1 will then assign T1, T2, T4 to itself, and send an assignment message for T3 to A2.
    A1,ASSIGNTASK,::A1:::::
    <T3>
16. In order to execute T1, T2, T4 A1 can now use the cached FSM and A* output from its earlier calculation during the negotiation phase. This schedule executes T2 first, then T1 and T4, with overall quality of 1118.
17. For T3, A2 has a schedule which yields a quality of 436.
18. After some time, when A1 has completed the first leg of its schedule and completed task T2. It will have removed T2 from its pending tasks list, and its physical location is now at the old location of T2, going towards T1.
19. Similarly, A2 will have completed T3 and is stopped at its location.

20. At this time, we inject two new tasks into the system, which are shown in the figure below, as T5 and T6. These are shown in the figure below:

Figure 10 Additional injected tasks and new agent locations

21. A1 again receives an external message via MQTT which is

    A1,NEGOTIATE,::::-T5-T6-

    <>

    A1 will first calculate the incremental quality of executing these itself, and then ask the managed agents to report theirs.

22. A1 first calculates the schedule quality of its pending tasks T1 and T4, by repeating steps 5 to 8. The quality comes to 638, which acts as the base quality of current schedule.

23. A1 then creates a power set of all currently pending and newly injected tasks, which includes: {T5}, {T5, T4}, {T5, T1}, {T5, T1, T4}, {T5, T6}, {T5, T6, T4}, {T5, T6, T1}, {T5, T6, T1, T4}, {T6}, {T6, T4}, {T6, T1}, {T6, T1, T4}, {T1}, {T1, T4}, {T4}

24. These tasks are then expanded into FSMs following the Taems QAFs, one being shown below.

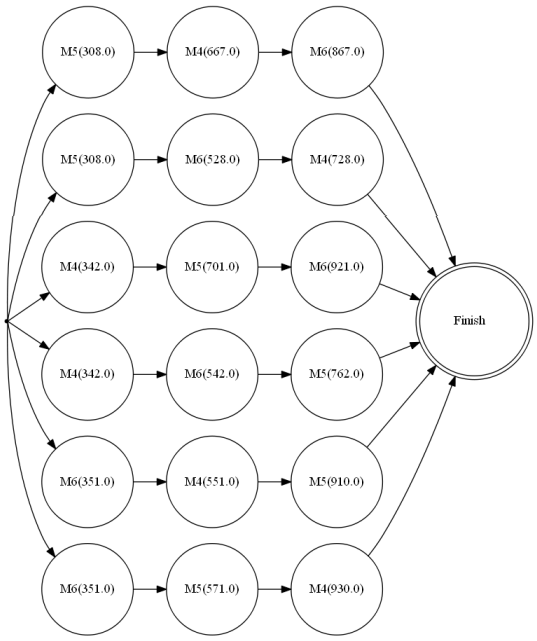


Figure 11 Expanded FSM for step twenty six

25. A* algorithm is then applied to find the best route qualities from all of the expanded FSMs. In this case, we note that the base quality is not zero, but has a value.

    [T1,638,388]

    [T1-T6,638,738]

    [T1-T4,638,638]

    [T1-T4-T6,638,951]

    [T1-T5,638,584]

    [T1-T5-T6,638,958]

    [T1-T5-T4,638,997]

    [T1-T5-T4-T6,638,1317]

    [T5,638,308]

    [T5-T6,638,571]

    [T5-T4,638,701]

    [T5-T4-T6,638,930]

    [T4,638,342]

[T4-T6,638,551]

[T6,638,351]

26. It then sends a message to its managed agent, A2, to report its cost calculation

    A2,CALCULATECOST,::A2:::::A1

    <T5,T6>

27. A2 combines the new pending tasks T5 and T6, with its current schedule which happens to be empty because its only task previously assigned to it, T3, is already complete. A2 goes through the same steps at #5 to #8 and then sends back the following message:

    A1,COSTBROADCAST,::A1::-T1-T4-T5-T6-:::A2

    <[T1,0,258]

    [T1-T6,0,685]

    [T1-T4,0,535]

    [T1-T4-T6,0,935]

    [T1-T5,0,566]

    [T1-T5-T6,0,940]

    [T1-T5-T4,0,979]

    [T1-T5-T4-T6,0,1329]

    [T5,0,370]

    [T5-T6,0,590]

    [T5-T4,0,729]

    [T5-T4-T6,0,929]

    [T4,0,285]

    [T4-T6,0,535]

    [T6,0,335]>

28. After receiving this message, A1's data collection is complete, and it can now apply the SAT Solver to find out the new best allocation. First, the OPB format input mapping is created, as follows:

    * #variable= 30 #constraint= 6

min: -388 x1 -738 x2 -638 x3 -951 x4 -584 x5 -958 x6 -997 x7 -1317 x8 -308 x9 -571 x10 -701 x11 -930 x12 -342 x13 -551 x14 -351 x15 -258 x16 -685 x17 -535 x18 -935 x19 -566 x20 -940 x21 -979 x22 -1329 x23 -370 x24 -590 x25 -729 x26 -929 x27 -285 x28 -535 x29 -335 x30;

* x1=1[1] x2=1[1, 6] x3=1[1, 4] x4=1[1, 4, 6] x5=1[1, 5] x6=1[1, 5, 6] x7=1[1, 5, 4] x8=1[1, 5, 4, 6] x9=1[5] x10=1[5, 6] x11=1[5, 4] x12=1[5, 4, 6] x13=1[4] x14=1[4, 6] x15=1[6] x16=2[1] x17=2[1, 6] x18=2[1, 4] x19=2[1, 4, 6] x20=2[1, 5] x21=2[1, 5, 6] x22=2[1, 5, 4] x23=2[1, 5, 4, 6] x24=2[5] x25=2[5, 6] x26=2[5, 4] x27=2[5, 4, 6] x28=2[4] x29=2[4, 6] x30=2[6]

1 x1 1 x2 1 x3 1 x4 1 x5 1 x6 1 x7 1 x8 1 x16 1 x17 1 x18 1 x19 1 x20 1 x21 1 x22 1 x23 = 1;

1 x3 1 x4 1 x7 1 x8 1 x11 1 x12 1 x13 1 x14 1 x18 1 x19 1 x22 1 x23 1 x26 1 x27 1 x28 1 x29 = 1;

1 x5 1 x6 1 x7 1 x8 1 x9 1 x10 1 x11 1 x12 1 x20 1 x21 1 x22 1 x23 1 x24 1 x25 1 x26 1 x27 = 1;

1 x2 1 x4 1 x6 1 x8 1 x10 1 x12 1 x14 1 x15 1 x17 1 x19 1 x21 1 x23 1 x25 1 x27 1 x29 1 x30 = 1;

1 x1 1 x2 1 x3 1 x4 1 x5 1 x6 1 x7 1 x8 1 x9 1 x10 1 x11 1 x12 1 x13 1 x14 1 x15 = 1;

1 x16 1 x17 1 x18 1 x19 1 x20 1 x21 1 x22 1 x23 1 x24 1 x25 1 x26 1 x27 1 x28 1 x29 1 x30 = 1;

29. The solver returns the following output:

    [-1, 2, -3, -4, -5, -6, -7, -8, -9, -10, -11, -12, -13, -14, -15, -16, -17, -18, -19, -20, -21, -22, -23, -24, -25, 26, -27, -28, -29, -30]

    Which indicates that assignment represented by variables x2 and x26 is the best.

30. Based on this output, A1 assigns T1 and T6 to itself and sends the following assignment message to A2 to assign T4 and T5 to it.

    A1,ASSIGNTASK,::A1:::::

    <T4-T5>

31. A1 executes its assigned tasks according to the schedule calculated earlier, T1 first followed by T6, with accrued quality of 738.
32. A2 executes its assigned tasks according to schedule T5 first and T4 second, with quality 729.

5. EVALUATION & EXPERIMENTAL RESULTS

Our solution for finding an optimal schedule relies on the SAT solver. These are known to be NP-Complete, although many efficient implementations exit that use heuristics based solutions.

In order to compare the computational benefit obtained by transforming our problem into a SAT problem, we compared it with a brute force algorithm to find the best scheduling solution. This algorithm works as follows:

```
Input: Agents, Tasks, List<AgentUtilities>
AgentsSize <- Size(Agents)
TaskCombinationSize = Size(List<AgentUtilities)
bestAllocation <= null
bestScheduleUtility = 0
Do for i=0 to TaskCombinationSize ^ AgentsSize
  allocation <- GetAllocationForVariable(i)
  If IsValid( allocation )
    utility = GetUtility( allocation )
    if utility > bestScheduleUtility
       bestAllocation = allocation
       bestUtility = utility
return bestAllocation
```

This algorithm is exponential with respect to number of tasks and agents. Some computational optimization has been done by exiting the loop for those cases which are clearly known to be infeasible, e.g. two agents performing the same task. But for all other cases, the brute force algorithm considers all possible allocations, calculate the quality accrued with that allocation, and then finds the one with highest quality. This brute force

algorithm is always guaranteed to find the most optimal solution.

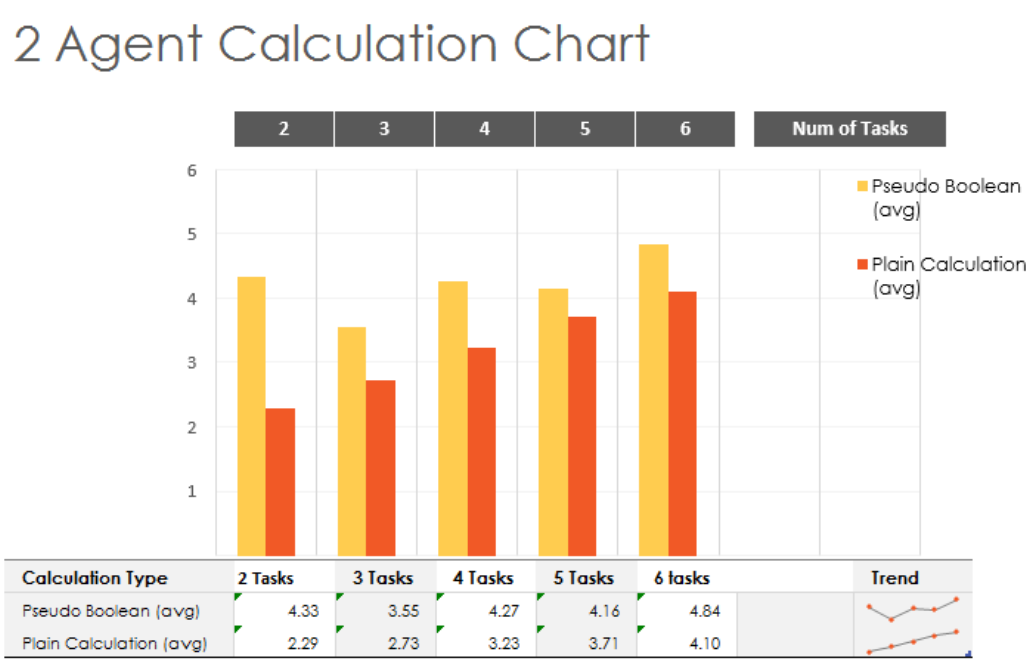
For our experimentation, we instrumented the code at the exact places where the allocation logic is entered and exited, and ran it twice one for the brute force algorithm and one for the SAT based algorithm, for the same problem. Both time values were calculated on the same machine and same memory and CPU environment.

We started with small values of agents and tasks (two agents, two tasks) and kept on increasing each separately, until we reached six agents and six tasks. We used a test harness to generate random location of tasks, and then fed those tasks into the system for it to calculate the utilities for each agent and finding the best solution. For experimentation, we modified two elements of our source code.

1. First, we added a separate step in the managing agent's schedule calculation logic, to also calculate the schedule, using same data, but by using the PlainCalculator class, instead of the PseudoBooleanCalculator class.
2. Second, we surrounded these calculation with timer functions, to capture the number of microseconds it took to perform the calculations.

The Pseudo Boolean Solver function calls had some fixed overhead related to file reading operation, and initializations of the solver, which had the potential of skewing the results. To avoid this, we calculated the time consumed when applying the solver to a completely trivial problem, involving a single task and two agents, averaged these values, and then subtracted this from all PB durations, to make sure that the time considered is only that of solving the actual problem at hand.

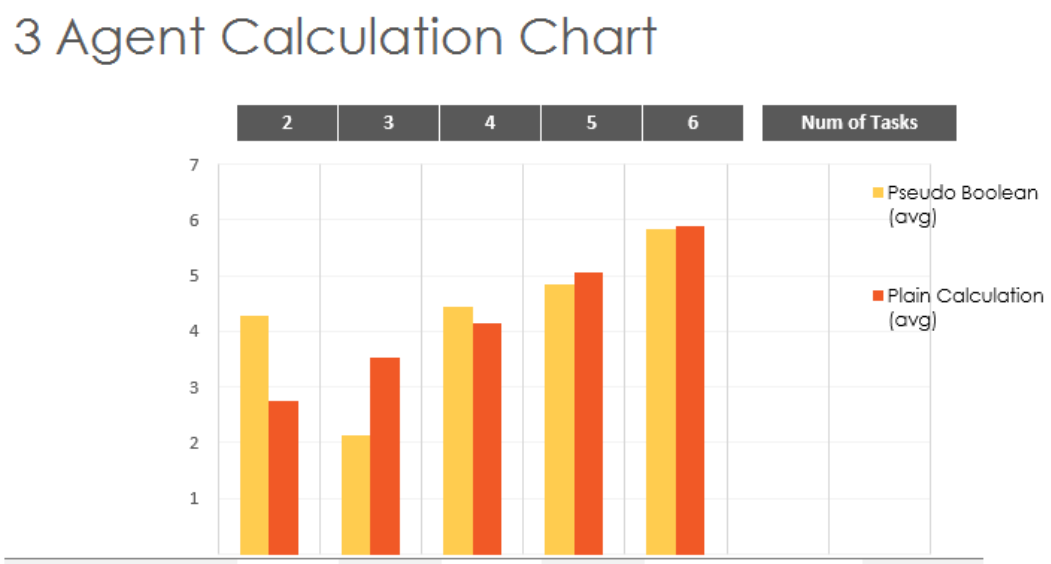
The results are provided in the charts below. Because of the vast differences in the time it took to perform the calculations, results are shown in logarithmic scale.



Calculation Type	2 Tasks	3 Tasks	4 Tasks	5 Tasks	6 tasks	Trend
Pseudo Boolean (avg)	4.33	3.55	4.27	4.16	4.84	
Plain Calculation (avg)	2.29	2.73	3.23	3.71	4.10	

Iteration #1 PB	128494	62138	67700	79221	135948
Iteration #2 PB	63283	70580	105112	75730	112749
Iteration #3 PB	54076	59516	64803	69927	140223
Iteration #1 Plain	201	577	1659	4377	12519
Iteration #2 Plain	198	495	1725	4893	11492
Iteration #3 Plain	181	533	1763	6085	13806
Control Case Average	60564				
	62328				
	61143				
	58221				

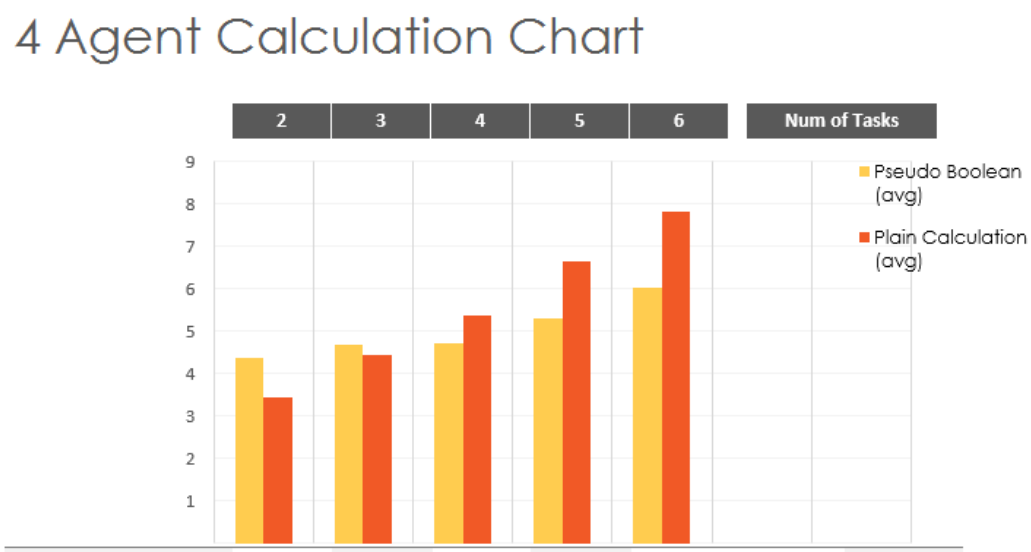
Figure 12 Two Agent calculations



Calculation Type	2 Tasks	3 Tasks	4 Tasks	5 Tasks	6 tasks	Trend
Pseudo Boolean (avg)	4.27	2.13	4.45	4.83	5.85	
Plain Calculation (avg)	2.74	3.53	4.15	5.04	5.89	

Iteration #1 PB	105570	57452	129112	144551	705624
Iteration #2 PB	72940	61814	71292	145598	767675
Iteration #3 PB	59600	62834	65150	94824	808545
Iteration #1 Plain	545	3467	14520	131019	731347
Iteration #2 Plain	557	3316	13327	99972	828636
Iteration #3 Plain	556	3415	14379	100218	745228
Control Case Average	60564				

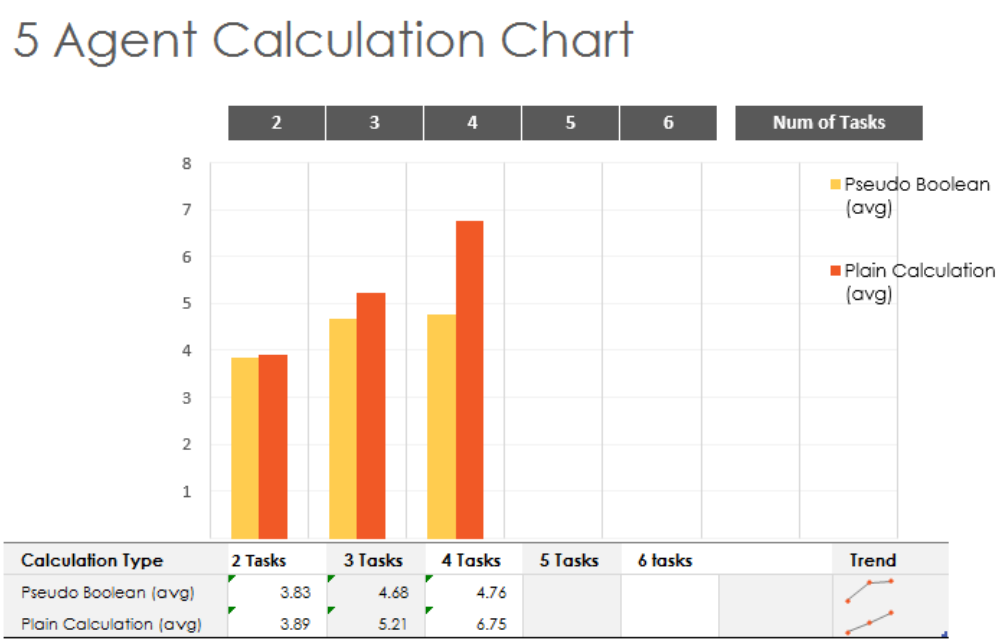
Figure 13 Three Agent calculations



Calculation Type	2 Tasks	3 Tasks	4 Tasks	5 Tasks	6 tasks	Trend
Pseudo Boolean (avg)	4.36	4.67	4.70	5.28	6.00	
Plain Calculation (avg)	3.44	4.44	5.35	6.65	7.80	

Iteration #1 PB	81190	123877	117888	298600	1068960
Iteration #2 PB	60718	75932	141909	198510	1065862
Iteration #3 PB	107827	122862	71675	252124	1078225
Iteration #1 Plain	2125	18582	249084	4378144	60878189
Iteration #2 Plain	1987	16798	202283	4575576	66261307
Iteration #3 Plain	4160	46749	213191	4533565	63231785
Control Case Average	60564				

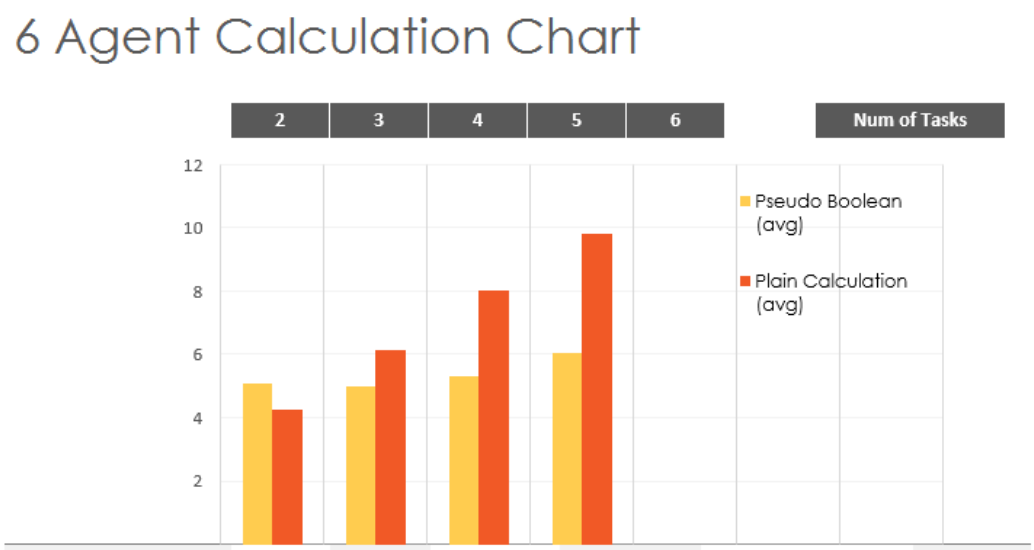
Figure 14 Four Agent calculations



Calculation Type	2 Tasks	3 Tasks	4 Tasks	5 Tasks	6 tasks	Trend
Pseudo Boolean (avg)	3.83	4.68	4.76			
Plain Calculation (avg)	3.89	5.21	6.75			

Iteration #1 PB	57287	69835	114006
Iteration #2 PB	84700	123845	141580
Iteration #3 PB	60139	131387	99551
Iteration #1 Plain	8104	114779	5605421
Iteration #2 Plain	10193	256492	5736107
Iteration #3 Plain	5190	116105	5452034
Control Case Average	60564		

Figure 15 Five Agent calculations



Calculation Type	2 Tasks	3 Tasks	4 Tasks	5 Tasks	6 tasks	Trend
Pseudo Boolean (avg)	5.08	4.99	5.30	6.03		
Plain Calculation (avg)	4.26	6.14	8.03	9.79		
PB Average	100928.67	117944.67	220,715.33	1098965		
Iteration #1 PB	142377	64840	164653	1131917		
Iteration #2 PB	59297	103425	268812	1066013		
Iteration #3 PB	101112	185569	228681			
Plain Average	18048	1386495	107541994	6223585410		
Iteration #1 Plain	16212	1366879	103266371	7215396234		
Iteration #2 Plain	18626	1238470	97569301	5231774585		
Iteration #3 Plain	19307	1554136	121790310			
Control Case Average	60564					

Figure 16 Six Agent calculations

A few things can be noted from these experimental results.

First we note that for very simple calculations, those involving just two or three agents, the plain calculation takes lesser duration than the SAT based calculation. This is true even after normalizing the calculation time by subtracting the time taken for the flat setup related time of the solver, which we had calculated earlier by passing a trivial problem through the same logic. In our mapping, the agents and tasks together become the number of variables, and the number of variables determines the calculation

complexity of the solver. Additionally, the number of tasks determine the constraints. Thus, for a very small number of variables and constraints, we see that we are better off deciding an allocation directly.

As we keep increasing the complexity, we eventually notice that the benefits of pseudo Boolean solver start showing, and the SAT based approach starts to complete in lesser duration for four agents and four tasks. Thereafter, SAT computation is always more efficient for any great number of tasks and agents.

We also note that as the complexity increases, for e.g. five agents and five tasks, the difference in computational efficiency becomes more and more pronounced.

The upper and lower bounds of SAT solvers range from $O(1.473^{a+t})$ to $O((a+t)^{1.801})$ (Biere 2009) where a is the number of agents and t is the number of possible task combinations. In our mapping of the problem to SAT solver, we can also specify a time value as parameter which determines the allowed time which it is given to run. The solver exists when that time elapses, and returns the best result it has found so far. If it is allowed to run continuously, it can take more time searching for an optimal solution, and if very little time is given, it is possible that it returns a sub optimal satisfying solution. However, the time it takes for brute force calculation to run, is usually much more than the time required to find an optimal solution. In our experiments, for the high complexity cases, e.g. 5 agents and 5 tasks, we found 25% cases at time limit of 1 second, where the results returned were not optimal. We can compare this with prior works e.g. (Modi 2005) where the authors assume associative aggregation functions. However, the approach does not guarantee optimality, although certain quality guarantees are still possible. In such approaches, the calculation complexity has been simplified and

delegated over to the agents, with the aggregation agent not required to perform any complex functions. Thus, our approach represents a different class of algorithms. Although our calculations are based on time, and their calculation is based on cycles, we can still see from the results that both algorithms manage to avoid exponential increase in calculation time, as demonstrated in Figure 9 in the paper.

Distributed agent based coordination has also been addressed using Max Sum algorithm (Farinelli 2014). These solutions rely on a domain specific characteristic that interactions between neighboring agents only are significant. E.g. when applied to the surveillance problem, the area between two agents need to be surveyed by any one of these two neighbors, and a third neighbor further down does not have any impact on quality accumulated by surveillance coverage in this area. MaxSum algorithm exploits the fact that some nodes (agents) need to pass messages between them, while others do not. The algorithm, falls back to its worst case performance, when all agents need to communicate with each other, which is the case in our example, where sequence and dependencies between tasks taken up by one agent can impact any other agent, which is implied by our support of the Taems scheduling structure. The complexity of that solution is thus governed by $O(t^{a+1})$, which is again an exponential bound, greater than the SAT based time bound.

6. FUTURE WORK

The work in this thesis can inspire of a number of avenues for future research. One aspect is improving the SAT mapping to not only include task allocation, but also task sequencing. Since the utility function in Taems is independent and domain specific, it might be possible to abstract out the utility function into its own set of variables, and form a Satisfiability problem whose solution partially contains elements of the task schedule itself. Task scheduling constraints are diverse and complicated enough that we do not expect a generalization to be available that can fully convert the problem into a SAT style input formulation. However, for certain vertical domains, it might be worthwhile to exploit common characteristics of the problem to transform into a pseudo Boolean optimization problem.

One other avenue for research is to use ADOPT (Modi 2005) based techniques to optimize the A* search, wherein the agent abandons exploration of a particular path based on knowledge which it has from previous executions and interactions with the managing agent.

In our A* algorithm, we also developed certain state space pruning strategies which include consideration of task deadlines to abandon exploration of certain sections of the generated FSM. However, based on Taems formalism, there are additional rules which lend this mechanism to be made more complete and comprehensive. This, however, is an improvement in the software, rather than a new academic research. Similar to this, additional software efficiency enhancements include some caching mechanism when applying A* search to find highest valued paths within the FSM.

The use of MQTT protocol in real life IoT situations e.g. drones or vehicular

agents, can be further studied to see what values of agent and task aggregation we well as Anytime characteristics of the A* search can be optimized to provide the most efficient execution.

Another direction for future research can be to apply incremental constraint optimization algorithms. For this problem, we also tried to combine inequalities where integers to represent decisions. However, we were limited by the tools available like GLPK, however, more general solutions based on this approach should be possible and is a good avenue for future research.

LITERATURE CITED

Berman, Spring, Vijay Kumar, and Adam Halasz. 2009. "Optimized Stochastic Policies for Task Allocation in Swarms of Robots." *IEEE*.

Biere, A., Heule, M., & van Maaren, H. (Eds.). 2009. *Handbook of satisfiability*. ios press.

Farinelli, A., Rogers, A., & Jennings, N. R. 2014. "Agent-based decentralised coordination for sensor networks using the max-sum algorithm." *Autonomous agents and multi-agent systems* 28(3), 337-380.

Fave, Delle, A Farinelli, R Jennings, and A Rogers. 2012. "A Methodology for Deploying the Max-Sum Algorithm and a Case Study on Unmanned Aerial Vehicles." *Association for Advancement of Artificial Intelligence*.

Horling, B., Lesser, V., Vincent, R., Wagner, T., Raja, A., Zhang, S., ... & Garvey, A. 1999. "The taems white paper."

Kim, Yoonheui, and Victor Lesser. 2013. "Improved Max Sum Algorithm for DCOP with n-ary Constraints." *Proceedings of the 12 International Conference on Autonomous Agents and Multiagent Systems.* Saint Paul: International Foundation for Autonomous Agents and Multiagent Systems.

Likhachev, Maxim, et al. 2005. "Anytime Dynamic A*: An Anytime, Replanning Algorithm." *ICAPS.*

Macarthur, Kathryn S, and et. al. 2011. "A Distributed Anytime Algorithm for Dynamic Task Allocation in Multi-Agent Systems." *AAAI Conference on Artificial Intelligence.* Southampton: University of Southampton.

Mailler, Roger T. 2004. *A mediation-based approach to cooperative, distributed problem solving.* Doctoral Dissertations Available from Proquest Paper AAI3136757, Paper AAI3136757.

Modi, P. J., Shen, W. M., Tambe, M., & Yokoo, M. 2005. "ADOPT: Asynchronous distributed constraint optimization with quality guarantees." *Artificial Intelligence* 161(1), 149-180.

Muscettola, Nicola, and et. al. 1998. "Remote Agent: to boldly go where no agent has gone before." *Artificial Intelligence 103.*

OASIS. 2014. *MQTT Specification.* October 29. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html.

Parrain, Daniel Le Berre and Anne. 2010. ". The Sat4j library, release 2.2." *Journal on Satisfiability, Boolean Modeling and Computation, Volume 7* 59-64.

Pujol, Marc. 2015. *JMaxSum.* https://github.com/RMASBench/jmaxsum.

Ramchurn, Sarvapali D et. al. 2010. "Decentralized Coordination in RoboCup Rescue." *The Computer Journal.*

Rayside, Derek, Estler Christian, and Daniel Jackson. 2009. *The Guided Improvement Algorithm for Exact, General Purpose, Many Objective Combinatorial Optimization.* Cambridge: Computer Science and Artificial Intelligence Laboratory, MIT.

Rayside, Derek, H-Christian Estler, and Daniel Jackson. 2009. *The guided improvement algorithm for exact, multi-purpose many-objective combinatorial optimization.* Cambridge: Massachusets institute of technology.

Roussel, Olivier. 2007. *PB06: Input Format.* http://www.cril.univ-artois.fr/PB07/pb06_inputFormat.html.

Ruiz, Esmeralda. 2011. "Distributed SAT." *Artificial Intelligence Review* (Artificial Intelligence Review) Volume 35, Issue 3 , pp 265-285.

Tambe, Milind, Ranjit Nair, and Makoto Yokoo. 2005. "Network Distributed POMDPs: A synthesis of Distributed Constraint Optimization and POMDPs." *American Association of Artificial Intelligence.*