אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

Faculty of Engineering Sciences

Department of Industrial Engineering & Management

Final Report

Applying Max-Sum Belief Propagation to Service-Oriented Multi-Agent Problems

2022-01-224

Submitting:

Tomer Kigel and Noy Kasher

Supervised by:

Prof. Roie Zivan

30.06.22

Abstract

Multi-Agent Optimization Problem (MAO) have many realistic applications in environments where autonomous entities cooperate to achieve a global objective. In a service-oriented MAO, the objective is to allocate service providing agents to service requiring tasks. Service providing agents decide to which of the service requiring agents to provide service and in what order. The utility derives depends on the quality of the service, the time it took to provide it and the cooperation among service providing agents. The solution for the problem is a schedule for each agent, with an aim to maximize the team's global utility.

A common method for solving such problems is by modeling them as Distributed Constraint Optimization Problems (DCOPs) and applying DCOP algorithms for solving them. However, the standard DCOP structure is not suitable for representing such problems due to their natural division between service-providers and service-requesters. Moreover, service-providers interact only with service-requesters and vice-versa, so the structure of the graph representing service-oriented problems is bipartite.

Transforming DCOPs to a bipartite graph, in which there is a clear distinction between two sets of nodes, will allow us to address the limitation of the standard DCOP model and run existing algorithms to solve service-oriented problems. "Max-Sum" is one such algorithm; it is the "belief propagation" version that solves DCOP, and it operates on a bipartite graph. This makes "Max-Sum" a natural choice for solving service-oriented multi-agent problems.

In our project, we focused on solving the problem mentioned above using "Max-Sum". The main challenge we encountered is symmetry; When multiple service-providers are required to fully serve a service-requester, due to the nature of "Max-sum", the common result is that all of the service-providers are being attracted to the service-requester, or none of them. We proposed heuristics to overcome this phenomenon. Previous work proposed the Service-Oriented Multi-Agent Optimization Problem (SOMAOP) model and algorithms for solving it. In our project, we applied "Max-Sum" to SOMAOP, and compared it with SOMAOP algorithms.
Keywords: Multi-Agent System; Task Allocation; Distributed Constraint Optimization Problems (DCOPs); Max-Sum.

Table Of Contents

1. Literary Review

1.1 Introduction

Distributed Constraint Optimization Problems (DCOPs) is an elegant model for representing and solving many real problems that are distributed by nature and cannot or should not be solved centrally. DCOPs are NP-hard and therefore, incomplete algorithms are considered for solving them. 'Max-sum' is one such algorithm, it is the 'belief propagation' version that solves DCOP. It operates on a factor-graph, which is a bipartite graph including variable nodes and function nodes. Service-oriented problems e.g., Task allocation problems, can be modeled as a DCOP even though they are not naturally DCOPS. That is because their natural structure is similar to a factor graph. This makes Max-Sum a natural choice for solving them. However, there are several challenges; The main challenge we will attempt to solve in this project is the challenge of symmetry – In each service allocation problem, service requesters are required to decide on a preference of their neighboring service providers and based on this preference to offer utility to the different providers in order to attract service. This preference as well as the offers are decided simultaneously for all providers and occur under optimistic assumptions due to the nature of Max-sum. A common result is - the requester sends similar messages to all of his neighbors asking for their service or fully denying the need for their service even though that strategy is counterproductive.

1.2 DCOP's

DCOPs are typically modeled as a network of agents each holding variables. Variables have domains that describe the possible values each variable can be assigned. Constraints are costs that are incurred as a result of combinations of values assignments to different variables and are held by agents. In a factor graph, the function nodes are special nodes (that can also be addressed as agents) designated for representing constraints. The communications between agents allow them to cooperate in the task of finding a globally optimal solution, concerning the costs of the constraints [6]. However, agents are unaware when a good global solution is obtained because of the difference between the global evaluation of the system's state and their private evaluation of states [1]. In this project, we will focus on Max-Sum, an inference incomplete algorithm for solving DCOP, and will apply it to service-oriented multi-agent problems.

1.3 Belief propagation

Max-sum is an asynchronous version of Belief Propagation, used for solving DCOPs. Belief Propagation is a message-passing algorithm for performing inference on graphical models. The algorithm works in the following manner: Each node collects messages from neighboring nodes in a graph, and then all messages but one are used as inputs to an internal function. The output of the function which is considered as a 'belief' is then being passed as a message to the node whose message has been ignored. Eventually, nodes perpetually update their own beliefs based on incoming messages from all of their neighbors and commonly converge to certain beliefs. The algorithm's strength is the rather fast convergence to satisfying solutions at the cost of sub-optimality [3].

1.4 Max-sum

Max-Sum is a completely exploitive algorithm that doesn't necessarily converge. On tree-structured graphs, max-sum converges to the optimal solution in linear time. On graphs with multiple cycles, there is no guarantee that it will converge; different solutions have been and are developed to make sure Max-sum produces viable results in those cases, for example, implicit exploration with 'anytime' feature can be used as a strategy.

Max sum is an **inference algorithm**. In contrast to search algorithms, which traverse the solution space directly by choosing different value assignments, inference algorithms aim to propagate information through the system that can be used by agents to select a high-quality solution.

Max-sum does it by propagating sums of costs/utilities and then inferring the value assignments that lead to the best costs/utilities [1]. In case there is a tie between two possible value assignments, the algorithm must employ a strategy for tie-breaking such as 'dust' or value propagation.

1.5 Dust and value propagation

'Dust' is the idea of adding small in-significant random values to the costs associated with each constraint, such that the probability to get equal costs using the algorithm will be infinitesimal. This method might not work with cyclic graphs. The way this method is implemented is by creating unary constraints for all agents with costs/utilities that are orders of magnitude smaller than the utility the agent can supply [5].

Another way to settle equality is **'value propagation'**. The idea is to choose an assignment between equal alternatives and propagate the selection to all nodes in the graph to infer all other

assignments. This method reduces the privacy of agents as they have to share their chosen values with their neighbors [9].

1.6 Parameters – Factor graphs

Max sum uses a factor graph model which has two types of nodes; **Variable nodes** are nodes that need their values to be inferred or decided, and **factor nodes** are nodes that hold the relationship between variables. Variable nodes and function nodes are considered "agents" in Max-sum, i.e., they can perform computations as well as send and receive messages. Messages are only sent from variable nodes to neighboring function nodes and from function nodes to neighboring variable nodes, and are called "query" and "response" respectively. "Query" messages are computed by summing all messages from the previous iteration that were received by a variable node. "Response" messages are selected outputs from a set of alternatives produced by an internal function mentioned thereafter.

Using factor graphs makes it easy to model many real-life problems. For instance, real-life problems tend to be asymmetric and factor graphs deal well with it since factor nodes can represent individual 'perceptions' of each agent on the world separately [13]. Also, many real-life problems have a nature of a dichotomy between service providers and events, that distinction fits factor graphs well.

1.7 Parameters - Target and interaction functions

As mentioned above, all messages collected by a single node in the graph are used as inputs to some function. That function is a composite of a basic mathematical operation such as summation or multiplication, and a process of selection. That operation must have a goal for it to improve over time. That goal is the target function and it is defined by the process of selection. Meaning that selecting which output of the internal function we want to keep and use and which to discard guides the process towards the desired outcome. Min-sum (which we refer to as Max-sum) strives to minimize the sum of constraint costs.

1.8 Transforming a DCOP to a factor graph

DCOP's can be easily transformed into factor graphs as follows; Variables are transformed to variable nodes while constraints between variables are transformed to function nodes. Agents are a mere abstract idea, and in the transformation, they are reduced to multiple variable nodes since each variable the agent holds becomes a standalone variable node [2]. By transforming DCOP's to

factor graphs, we can run Max-sum and other **service-oriented algorithms** that work strictly on factor graphs. Service-oriented algorithms have a clear distinction between agents representing service providers and service requesters, thus both of them can be mapped to variable nodes and function nodes respectively.

1.9 Types of graphs and convergence

'Max-sum' can run on different types of graphs that have inherently different properties. The two main types are trees and cyclic graphs.

1.9.1 Trees

When implemented in tree-structured problems, Max-Sum behaves similarly to dynamic programming in all directions; Each node in the graph is treated as though it is a root of its own graph. Each such graph with a unique root goes through a dynamic programming algorithm until the root holds an optimal solution. Eventually, all nodes hold a unanimous optimal answer to the whole problem. The conclusion is that Max-sum guarantees convergence to the optimal solution in linear time when the constraint graph (and as a result, the corresponding factor graph) is a tree graph [2], i.e., contains no cycles, so long so we know how to settle cases of equality.

1.9.2 Cyclic graphs

Once a graph includes **cycles**, it is no longer a tree and thus major problems are introduced; The primary problem is that Max-sum has **no guarantee to converge** [10,11]**.** Another problem in cyclic graphs is the exponential growth of messages, which is the multiplication of the content (values) carried by the messages. At intersections with more than two neighbors, the original message is sent to multiple recipients, meaning the content the message is carrying will be summed up more than once in multiple different places. As a result, the calculation becomes **inconsistent,** which in turn leads the Max-Sum algorithm to explore low-quality solutions.

A single cycle graph is a unique case in which exponential growth of messages and inconsistency doesn't occur, therefore it may converge. A single cycle graph can be thought of as an infinite chain of nodes, and therefore if the combination of assignments of all agents in the graph ever repeats itself, we know the algorithm has converged.

1.10 Improvements

To address the limitations mentioned above and to increase the possibility that the algorithm will converge, **damping** was suggested - giving more weight to new data and prioritizing most recent events. However, damping slows down the propagation of information among agents, and therefore, to shorten the process of producing high-quality solutions by Max-Sum, **splitting** a factor node into two new function nodes that together hold the original constraints was proposed. Splitting counteracts damping by producing more messages in each iteration which mitigates the effects of damping. The approach that combines damping and splitting produces high-quality results [2].

1.11 Task allocation – Service oriented problems

In the real world, there are many problems that do not look anything like a DCOP yet they can be easily modeled in factor graphs and be solved as if they were DCOP's by using one of the many known algorithms used for solving DCOP's. Those problems are service-oriented, task allocation problems. In these problems, the objective is to allocate service providers to service requesters in the best way possible, meaning, with the highest utilities or lowest costs of the agents arriving at tasks. The solution for the problem is a schedule for each provider, containing the order of the tasks to be performed. Mapping service providers to variable nodes and service receivers to factor nodes on a factor graph allows us to run Max-sum on a service-oriented problem. This structure represents the provider-requester scheme best, as in realistic applications, the requesters will interact with providers and vice versa.

In these problems, each service requester tends to have a utility budget it can use to attract service providers. This utility budget is the mechanism through which service requesters decide what providers they prefer to provide service to.

There are parameters that are unique to service allocation problems, and they define the utility gained from service providers. These parameters define the ‘capability’ of a task, meaning these are the requirements of the task which affect its completion [7]; **Distance** is related to the time of service provision, as giving service too late or too early might not produce any utility (i.e. someone in need of medical attention dies due to lack of treatment). **Skills** are the different abilities each service-provider has. The utility they can offer to a service-requester depends on the needs of the requester. **Quality** of service is a parameter affecting the utility received by the service, as higher quality would often translate to higher utility. Finally, limitations of the **number of providers** per

task dictate whether coalitions can or cannot be formed for the completion of tasks and their size. **Coalitions** can improve the efficiency of the task completion, raising the capability measure of the task [8].

1.12 Our goal and our challenge

Our work will focus on solving service-oriented, task allocation problems using 'Max-sum' as efficiently as possible. The challenge we expect to encounter is the problem with symmetry. When multiple service providers are required to fully serve a given service requester an issue arises.

A service requester will receive messages from each service provider they are neighboring and will make calculations about who should they request service from. In the 'Max-sum' algorithm, these calculations assume all agents will behave optimally. So, if a subset of neighbors can fully provide the service to the requester, the algorithm will send messages to the providers that are not a part of the subset that their services are not required.

For each service provider, a calculation process takes place where the main question at hand is whether a subset of service providers exists such that they fully serve the requester without the participation of the service provider we are calculating for. Since it is possible that a subset of that type exists, the end conclusion, in that case, will be the service provider for which we make the calculation is not required. Since the process repeats for all service providers the result is all providers might believe their services are unneeded. This result is an unwanted behavior. In addition, even if not all providers believe they are unneeded, the requests that will be sent out are not guaranteed to be sufficient to attract enough service providers to fully serve the requester. The paragraph above addresses the symmetry problem we have described in our introduction and is the focal point of our paper.

1.13 Known solutions to the problems of symmetry

The first solution to the problem of symmetry is the idea of '**Function Meta Reasoning**' (FMR). The concept is to choose a heuristic by which we will take a set of service providers neighboring a service requester and prune it such that we will be left with a subset of service providers that can only fully serve the requester if all of them agree to cooperate. We will update the original problem graph such that the neighbors that are not part of the new subset will be detached from the service requester. Since this newly constructed subset can't guarantee full service without all participants taking part, 'Max-sum' won't default to releasing service providers from duty as previously stated

[5]. The purpose of such manipulation is to incentivize exploration for the detached neighbors and to break the symmetry.

The second solution which is a supplement to the first is **'Ordered Value Propagation'** (OVP). Once a pruned subset is produced by the FMR, the subset of selected agents may hold collectively a total utility that exceeds the amount needed for the service requester. That fact maintains the problem of symmetry, thus we would like to reduce the utility of providers such that the total utility won't exceed that exact required amount. We can achieve this in multiple ways; one way is to select a single provider and reduce his utility by the required amount. Another way is to reduce the utility of all agents by a fixed amount equal to the difference between the existing utility and the required utility divided by the number of agents in the subset. [4,5].

2. Project Planning

2.1 Introduction to the Service Oriented Multi-Agent Optimization Problem (SOMAOP)

The distinguishing feature of the service-oriented problems is the natural division between agents providing services and agents requesting services, where the providers only interact with the requesters and vice-versa.

Previous work in the department proposed an abstract model to better represent the variety of problems that fall into that category, i.e., Multi-Agent Optimization Problems (MAO) in which there is a clear distinction between two sets of agents. A Service Oriented Multi-Agent Optimization Problem (SOMAOP) is a system with agents representing service providers (SP) and agents representing service requesters (SR). Each SP has a set of skills that can be used to provide services, and a set of SRs that it can give its services to. Mutually, Each SR has a certain set of skills required to the task, and a set of SPs in its vicinity that can provide the service it needs. There is a local incentive to initiate service in order to gain utility; each SP receives payment for the service it provides, and each SR receives a benefit from the service it needs. The global incentive is to maximize utility to the SRs of an allocation. The structure of SOMAO is bipartite, as agents from one set only interact with agents of the other set, therefore it represents the provider-requester scheme best. The solution for the problem is a tasks schedule for each SP.

2.2 Simulator

Before our involvement, to test the theoretical model a simulator was created. The simulator is abstract by nature and was designed to simulate an environment of service-oriented problems which can be solved by the implementation of different distributed algorithms. It receives input parameters for generating a SOMAOP as well as the algorithm by which we want to solve the problem, and it outputs the solution found. As for now, the simulator can run multiple algorithms which are written in a distributed and synchronized settings, as well as compare their outcomes for the same initial SOMAOP.

2.3 Visualization

Firstly, we have visualized the SOMAOPs to better understand them via Python libraries. The graph attached in figure 1 shows a service-oriented problem that was drawn by the abstract simulator. Squares represent SPs, while circles represent SRs. The agents were randomly positioned and their

locations in the graphic display are significant since the relative distances between agents can be deduced from them. Edges connecting agents represent adjacency, where SP can be adjacent only to SR and vice-versa, as befits to a bipartite graph. Moreover, adjacency exists if SP is at a reasonable distance (pre-defined) from SR and if it has the required set of skills to perform the task.

Finally, red edges represent a partial allocation in a particular iteration of a certain algorithm implemented in the simulator. The green color of SRs indicates the extent to which service was provided by SPs relative to the maximum service required by the given SR; dark green vertices indicate fully served SRs, while completely white vertices indicate SRs who didn't received service at all.

Importantly, a feasible allocation is a solution where each SP has an assignment to a single SR at a time, with the aim of maximizing the global utility of the combination of all the SRs assignments.

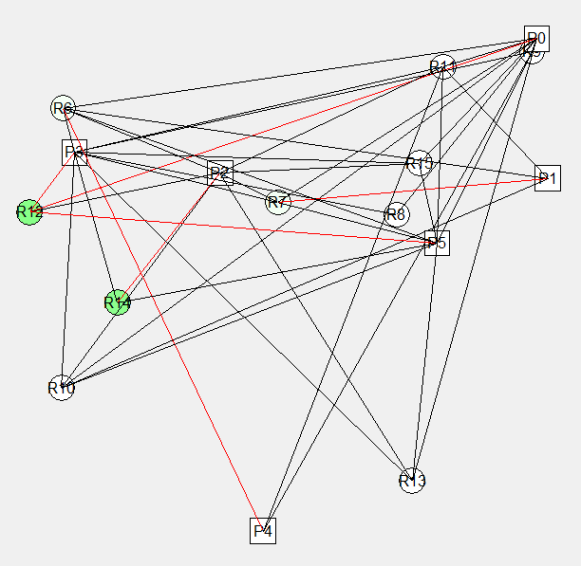


Figure 1: Visualization of a SOMAOP

2.4 Applying Max-Sum to SOMAOP

Our involvement is reflected in the implementation of the Max-Sum algorithm in the available abstract simulator with the addition of known improvements to help it diverge, as well as tackling expected challenges that are due to arise.

We have implemented a distributed environment that allows message passing between agents as a necessary preparation, and finally, we evaluated the behavior of Max-Sum when solving a SOMAOP, a combination that has not been investigated before, by comparing the solution produced by Max-

Sum against known solutions of other algorithms implemented in the simulator on the exact same problems.

Max-sum (node n)
1. $N_n \leftarrow$ all of n's neighboring nodes
2. **while** (no termination condition is met)
3. collect messages from N_n
4. **for each** $n' \in N_n$
5. **if** (n is a variable-node)
6. produce message $m_{n'}$ using messages from $N_n \setminus \{n'\}$
7. **if** (n is a function-node)
8. produce message $m_{n'}$ using constraint and messages from $N_n \setminus \{n'\}$
9. send $m_{n'}$ to n'

Figure 2: Standard Max-Sum

2.5 Significant challenges

2.5.1 Symmetry

The challenge we are facing up against is symmetry; when multiple service providers are required to fully serve a given service requester, an issue arises. In our problem, each service requester decides which service providers to attract. This decision is made simultaneously for all service providers and occurs under optimistic assumptions due to the nature of Max-Sum. A common result is – the service requester sends similar messages to all his neighboring service providers, asking for their service, or fully denying the need for their service even though that strategy is counterproductive. The direct consequence of the symmetry is that Max-Sum produces low-quality solutions, thus our aim is to overcome this challenge by implementing known solutions to the problem of symmetry; FMR and OVP.

2.5.2 Exploration

As previously stated, Max-Sum is a purely exploitive algorithm, both in the computation of its beliefs and in its selection of assignments based on those beliefs, thus we expect it to produce a straight monotonous rather than a scattered graph. The reason being is that when symmetry exists Max-Sum produces low-quality solutions, and once symmetry is resolved the utility propagated between agents doesn't change over time, it stays constant, and Max-Sum doesn't offer any exploration. Therefore, the second challenge we are facing up against is coming up with some creative options

to address this limitation and allow Max-Sum to become at the very least some-what explorative as it should be.

2.6 Assumptions

- SOMAOP is in its nature a dynamic model, meaning that there are changes that may occur over time. These changes are the disappearance and appearance of the agents in the system – SPs and SRs. However, for simplicity, we aim to solve a **static** problem, meaning that dynamic changes consist of the addition or the removal of agents cannot occur and are not considered during the run of Max-Sum.
- The solution for the problem involves a many-to-one structure, as in SOMAOP each SP can only be tending to one SR at a time. Therefore, the output is a **schedule** for each SP; each SP decides to which of the SRs to provide service and in what order.
- For Max-Sum to produce a schedule, we can choose one of two strategies: **One-Shot Max-Sum** or **incremental Max-Sum**.
- SPs are not allowed to **abandon** a task before finishing it.
- In One-Shot Max-Sum each SP "spends" all his skills for a certain SR, with no consideration of what skills are needed. Importantly, a SP won't receive a payment, i.e., a utility, for providing unnecessary skills.
- In One-Shot Max-Sum there is no consideration of the **coalitions** formed by the SPs during the run of Max-Sum, but during the calculation of the final utility of a certain allocation.

3. Project implementation and results

3.1 Implementation of Max-Sum

The Max-sum algorithm offers an innovative approach for solving DCOPs. Unfortunately, its main drawback is that is it a very exploitive algorithm. In SOMAOP Max-Sum is expected to converge to a single solution very quickly and stay there for the whole run of the algorithm. That is even after symmetry being addressed. To make Max-Sum more explorative a stochastic element can be added but it may still lead to exploration of poor results.

It was previously shown that when Max-Sum traverses poor results over time, **Damping** could be deployed to mitigate the effects of previously constructed beliefs that compound one on top of the other and allow for recent data to have more impact. That can lead to improved performance. Thus, we attempted to implement all of the ideas stated above, a solution to symmetry, adding a stochastic element and eventually damping.

3.2 One-Shot Max-Sum Algorithm for solving SOMAOP

According to this approach, the problem can be modeled such that each agent holds one variable exclusively which represents its schedule. It follows that the domain of each agent is immense due to number of feasible combinations of schedules it contains, what leads to exponential number of calculations. Therefore, we have set an upper bound to the number of permutations tested to reduce the domain.

Moreover, the skills (services) each SP provides to a certain SR are provided **together**. That is, each SP provides all the services it can provide while some may be unneeded. On one hand, a SP won't receive a benefit (utility) for a service that is not required, on the other hand, providing an unnecessary service will harm the overall utility as it wastes time that could have been invested in providing a necessary service to another SR.

3.3 Incremental Max-Sum Algorithm for solving SOMAOP

In contrast to the previous approach whereby each agent holds one variable which represents a combination of all agents' decisions, in this case each agent holds one variable for each decision it has to make. In other words, each SP has to decide who is the first SR to provide its services to, who is the second one, etc., with each decision depends on previously made decisions. Ultimately, Max-

Sum runs several times, and at the end of each increment the graph represents the problem changes to the new state, where the next decision has to be made.

In addition, the skills (services) each SP provides to a certain SR are provided **separately**. Therefore, unlike One-Shot approach, there is no such scenario in which an unneeded service is provided to a certain SR. That will allow the SP to move forward to the next SR and save time, and it is expected to have a positive effect on the global utility.

3.4 Utility Function

For comparing one algorithm to another we had to be consistent in the use of our utility function. In fact, the utility function is not part of the algorithm but a part of the definition of the problem.

That function addresses a maximum capacity parameter which limits the utility gain when multiple agents are present and giving service at the same time. This fact addresses implicitly OVP. By using the utility function in our calculations, we could avoid implementing OVP by ourselves as it was already doing it for us.

Moreover, the utility function rewards cooperation of agents so long so they don't exceed the maximal capacity. That means that the best algorithm for maximizing the utility would have to address cooperation between service providers in some way or form.

3.5 Challenges

3.5.1 NCLO

One iteration of Max-Sum is not equal to one iteration of another algorithm with a different runtime. In other words, the rate of advancement of each algorithm is different and therefore, it is not possible to compare the performance of differing algorithms by the criterion of iterations. To compare Max-Sum against other distributed algorithms implemented in the simulator for solving the same problem, we would like to define a unit of action that will be common to them; NCLO – the number of operations performed in a single iteration. Max-Sum is run synchronously so the amount of NCLOs is equal to the largest calculation done in a certain iteration.

3.5.2 Computational load

Max-Sum is a computationally demanding algorithm. Similar to dynamic programming, in every iteration a table is being constructed for each service requester after receiving messages from neighboring agents. The size of the table is typically a formula of variations with repetition as such:

$$\boldsymbol{amount\ of\ variations} = \boldsymbol{r^p}$$

In the incremental version of Max-Sum we implemented the algorithm to make decisions at a low resolution. Meaning the decision made in each increment is about where should a provider discharge a single unit of skill. Therefore, the decision an agent makes is not just a decision about what agent he should visit next. The decision is about the skill that should be supplied for the chosen service requester as well. Therefore, the table size increases in size even further depending on the number of skills each provider has.

It is easy to see the number of computations needed to run the algorithm is rather huge. To reduce the load of computations we use a simple but effective strategy to cut down the size of the tables we produce and thus reduce the number of calculations needed.

The first thing we do is treat assignments as binary options, either a provider is coming to a requester, or he doesn't. This reduction alone will cause the table size to drop from the original formula to:

$$\boldsymbol{amount\ of\ variations} = \boldsymbol{2^p}$$

This table size could get large yet still, but it is more manageable (especially if FMR is deployed). Now all that is left is to prove that we can use a table this size to achieve the same thing as with a table of the original size.

The second thing we do is constructing N tables of size $\boldsymbol{2^p}$. N being the number of skills the requester has. Each table will be associated with a single skill ID from the skill set the requester has. The reason for that will be shortly explained.

Additionally, it is important to note that the tables we build have to be built for each service provider separately, where in each table the relevant service provider is absent from the columns, meaning from the possible assignment. That is because Max-Sum makes decisions based on a partial

assignment for each service provider. That fact though does not affect the computational load very much.

Now every such table is built under the **assumptions** that there are only binary assignments where binary '1' means a provider will come with a single skill unit corresponding to the skill ID associated with the table, while a binary '0' means the provider won't come, but moreover it would mean the provider will necessarily go to a requester which he believes will benefit him the most (the requester which has the highest value in the belief vector the provider holds). We believe these two assumptions maintain the essence and properties of the original Max-Sum while allowing us to compute in reasonable times with and without FMR.

The justification is that Max-Sum is always prioritizing the maximal value in the table. So, as long as we maintain keeping rows that ought to hold the maximum value, we can reduce the number of needed calculations without compromises. Both our assumptions do exactly that.

Assignments that are labeled '1' would always be maximizing the cost of constraints, while assignments that are labeled '0' would always be prioritizing maximal belief values. All other assignments that were lost in this transformation couldn't ever have maximal values so long so the constraint and belief values can't be negative.

Lastly, the reason we have a number of tables that is equal to the number of skills a single requester has, is solely for our first assumption regarding the binary option '1'. This is because we only care for providers cooperating on the same skill for a given requester. Cooperation on different skills for the same requester is not expressed in some reward of utility, thus our assumption should hold true. Finally, we arrived to the following formula for calculating the amount of calculations for each requester:

$$\boldsymbol{amount\ of\ calculations\ for\ each\ requester} = \boldsymbol{P} * \boldsymbol{N} * \boldsymbol{2^p}$$

At the beginning when running the algorithm, the required number of operations is very high. Yet with every increment skills are being depleted, which leads to the reduction in size of the tables constructed. In sparse problems the effect is immediate and after the first increment the number

of operations required drops drastically. This behavior leads to a computational imbalance between iterations. The first iterations are more computationally demanding compared to the last iterations.

In our **One-Shot implementation** we made an assumption that in retrospect was probably faulty. We made the assumption the maximal value will always be produced at the row in the table representing an assignment where all providers are arriving to the given requester. Meaning that the size of the table for each requester is 1, therefore the NCLO is 1.
We did not take in consideration beliefs properly and we did not stick to the original Max-Sum algorithm. So computationally, running One-shot was very cheap and the results were not bad, yet it was not actually an implementation of Max-Sum but rather something else that was inspired by it.

3.5.3 Symmetry

3.5.3.1 FMR heuristic

The first solution to the problem of symmetry is "Function Meta Reasoning" (FMR). The concept is to choose a heuristic for renovation of the factor graph that represents the problem in a way that will result in adjacency between SPs and SRs that is defined by the needs of the SRs in the following manner: each SR will remain adjacent to a subset of SPs such that all of the SPs can give exactly full service to the selected SR if and only if all of them will decide to give service to the given SR together. All other SPs that aren't part of the subset will be detached in the graph. The purpose of such manipulation is to break symmetry. The selection of a subset of SPs who will provide service to a certain SR is done by sorting agents by distance and then going from the closest neighbor to the furthest adding each SP to a vector if and only if the contribution of an SP is larger than 0 when paired with all other SPs already in the vector.

In the incremental version of Max-Sum we work at a low resolution, we decide where a SP should provider a single unit of skill. Therefore, the FMR relates to every skill separately, that is because every skill has its own maximal capacity. So, in that case FMR does not actually renovate the graphs we work on per se. Instead, when building the tables described in the 'Computational load' section, each table associated with a different skill ID is getting reduced via the FMR logic. The number of columns which represent the different service providers is getting reduced to the bare minimum

needed to supply full service to the given SR. That is if and only if all of the providers left in the columns will cooperate.

In the One-Shot version we maintain the original idea of FMR where we actually renovate the graph and change the adjacency between SRs and SPs as described above.

3.5.3.2 OVP

The second solution to the problem of symmetry, which is a supplement to the first one, is "Ordered Value Propagation" (OVP).

In the One-shot solution, once a pruned subset of SPs is resulted by the FMR, the total utility provided by the agents may surpass the amount required by the SR and symmetry is maintained. Therefore, we are interested in closing the gap and reducing the collective utility provided by the SPs, so it won't exceed the required utility. we approached it by a balanced division i.e., we reduced the utility of each SP by a fixed amount equal to the difference between the provided utility and the required utility, divided by the number of SPs in the pruned subset.

In the Incremental version, The OVP is implied in the way we calculate the utility of service providers. Meaning the function that does the calculation of utility has a capacity parameter. When the amount of service provided exceeds it, the exceeding service will not count towards the utility returned by the function. Yet OVP still holds, just instead of it being balanced, the utility was always trimmed from the last agent that was added to the pruned subset.

3.5.4 Exploration

Once symmetry is resolved with FMR and OVP, Max-Sum converges extremely fast. The reason being is that agents do not propagate the relevant information and the effect of belief propagation is absent, so the product is that Max-Sum behaves like a local search. To allow Max-Sum to traverse the solution space and achieve balance between exploration and exploitation, we investigated adding a stochastic element:

- **Resetting the belief vector of a service provider** – each SP or SR owns a beliefs vector which indicates the payment for the service or the benefit from the service, as it believes, respectively. As the algorithm progresses, the gap between agent's beliefs grows and the

agent is fixed on a particular on a particular belief. As a result, when new messages arrive their contribution is negligible, so they unlikely to be able to change the agent's beliefs and the algorithm converges rapidly. To encourage exploration, we considered resetting an agent's beliefs vector with a small probability which would allow new messages to come into consideration and perhaps even change the agent's worldview. However, the effect of that heuristic is very similar to the effect of damping, as new messages have a significant weight in relation to old ones, so we decided to give up on this idea.

- **Confusion** – with a small probability of 0.3, a SR errs and sends to a SP erroneous message that were supposed to be sent to another SP, in order to add some noise to Max-Sum.
- **Skills Order –** each SP has a set of skills (services) that can be used to provide service. Initially, all SPs provided skills to the SRs in the same order, meaning there was a certain skill that always was provided late. Therefore, another stochastic element that was added to our algorithm is the order in which the skills are provided; each SP will provide its services to the SR in a different order that will be randomly drawn.

3.6 Final Decision Making

Eventually after building tables and making calculations for each table, together with FMR, OVP and adding a stochastic element to the equation we have to make a decision which message to send to all of the different service providers neighboring each service requester. In-order to achieve that we chose a single highest value row from each one of the N tables we have created. After doing that, we pick the one row that has the highest value among all of the selected rows. Finally, one we have a chosen single selected row representing an assignment we can make the final calculation for. That calculation is to check what is the additional utility the service provider brings to the partial assignment in the selected row. If the utility is larger than zero, which must be the case due to the function of FMR and OVP, the message produced will be that added utility value.

3.6.1 Building a schedule

To this point we have explained how Max-Sum functions in our implementations and how we have dealt with problems and challenges during implementation. Now we would like to explain how the final solution is being constructed.

In the incremental version we run the algorithm for a fixed number of iterations. Once it has finished, we chose the best result we were able to achieve by assuming we have anytime capability. Even though we didn't actually implement anytime in our virtual environment, we allowed ourselves to 'cheat' and reconstruct the best solution centrally. We can do that because for the purpose of research this decision has no impact on the results. In real life systems, anytime will have to be implemented. Once a solution is chosen, we advance the clock of each agent depending on the actions he performed and we add the chosen assignment to a list, the relevant service provider and service requester hold. Thus, all calculations on ward depend on previously chosen allocations. We repeat this process until reaching the iteration cap we have setup. In our experiments we have exhausted all resources (primarily skills and time) before reaching that cap value. Eventually the requesters and providers hold a full schedule which is their part of the solution to the problem.

In the One-shot version we have assumed service providers discharge all of their skills in once place (i.e., a service requester) so we run the algorithm similarly to the incremental version just without advancing time or making choices on top of other previously made choices. Instead, there is only one choice to be made per provider and that decision, once it is made, would be the partial solution to the problem. After we finish running the set number of iterations we use anytime just as before to reconstruct the best solution found.

Now that we have a schedule as a solution to the problem, we can easily compare our results to other algorithms and form an opinion as to the performance of Max-Sum in relationship to other alternatives.

3.7 Results

3.7.1 Experimental Setup

We ran Max-Sum Multiple times for each version we have described above. Our initial set up was the following:

- We had 10 service providers and 15 service requesters in each problem.
- The maximal amount of skill types each agent could have been 4.
- The Cap value was 2. (Maximal capacity per skill at a time)
- There was no distance limitation
- Maximal skill value per skill for requesters was 10.
- Maximal skill value per skill for providers was 3
- All agents were generated in a 50 by 50 square area. (Meaning maximal possible distance between 2 agents is approximately 70)

We ran the incremental version 200 iterations per problem. The One-shot version had 100 iterations instead.

In the incremental version there was a computational imbalance as we have discussed in this paper. The meaning of it was that calculations done before the first increment were plentiful while after the increment they got drastically reduced. So, we have decided to try and rebalance this effect by increasing the number of iterations we run our algorithm with each increment. The first increment got 20 iterations, and each subsequent increment got 10 more iterations compared to the previous one.

The stochastic element we have discussed previously was only introduced 5 iterations after each increment, that fact allowed Max-Sum to converge before we started introducing exploration. In the One-shot version we did the same thing.

Eventually, our different runs include the following final versions:

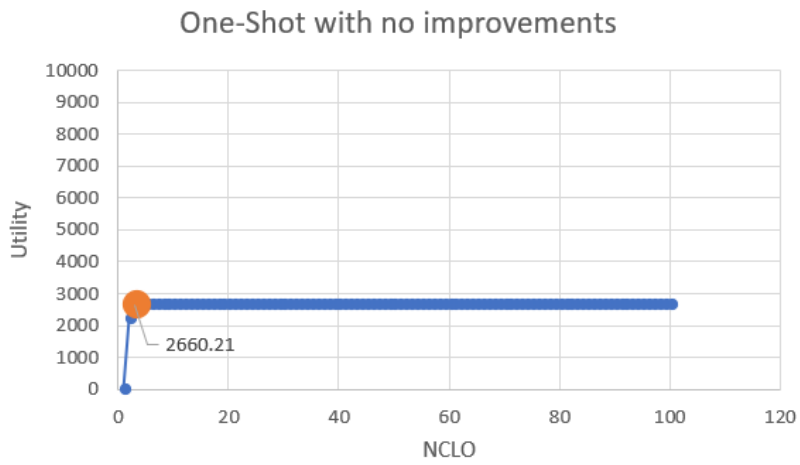
One-Shot:

- Vanilla One-shot Max-Sum (Without addressing the problems and challenges).
- A version where we implement FMR to address symmetry.
- Finally, a version with FMR with an added stochastic element to introduce exploration.

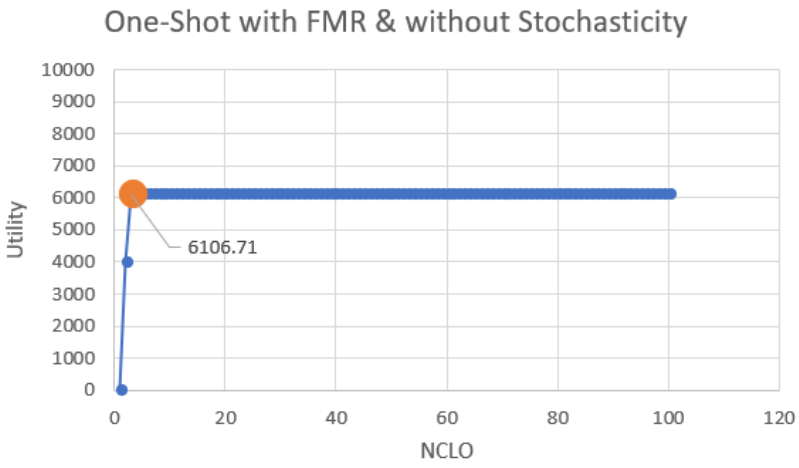
Incremental:

- Vanilla Incremental Max-Sum (Without addressing the problems and challenges).
- A version where we implement FMR to address symmetry.
- Finally, a version with FMR with an added stochastic element to introduce exploration.

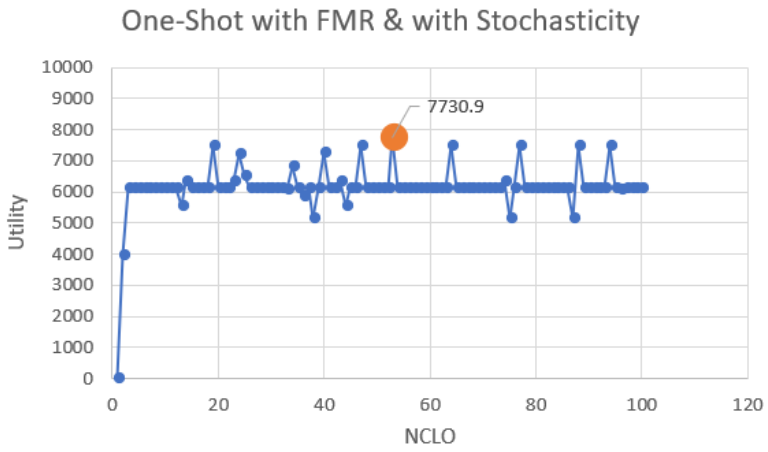
3.7.2 Single Run – One-Shot Max-Sum



In the graph above we see a single problem, a low utility of around 2500. It is expected to be low.

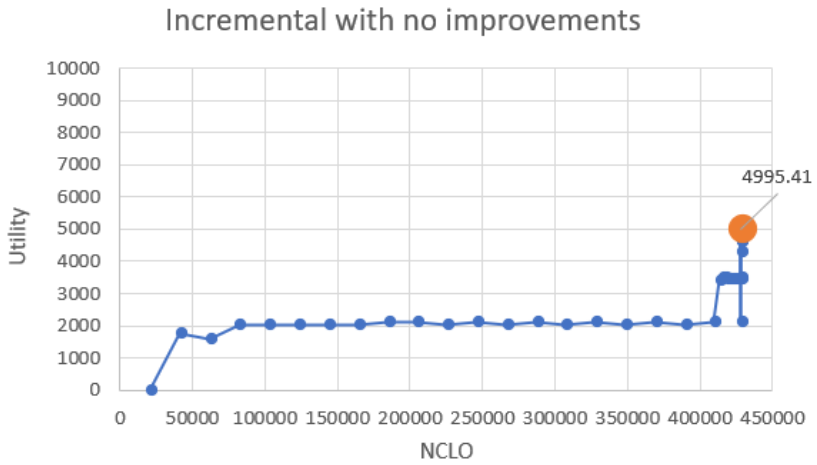


In the graph above we see the impact of adding FMR. The utility rises almost threefold. FMR is very effective.

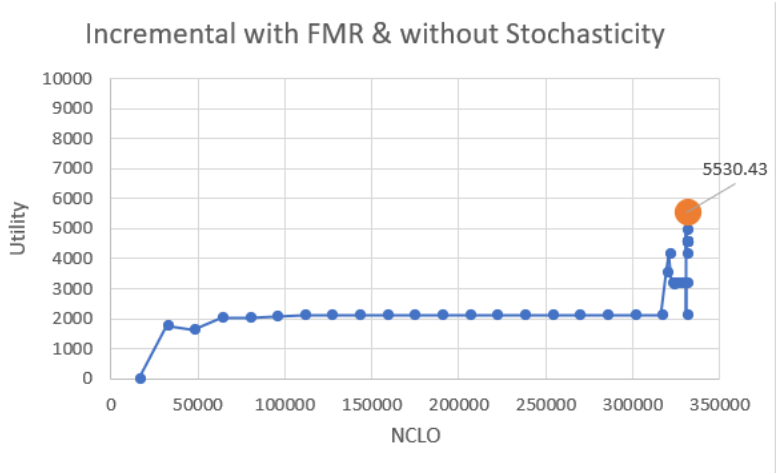


Finally in the graph above we see after adding the impact of some stochastic element after the implementation of FMR.

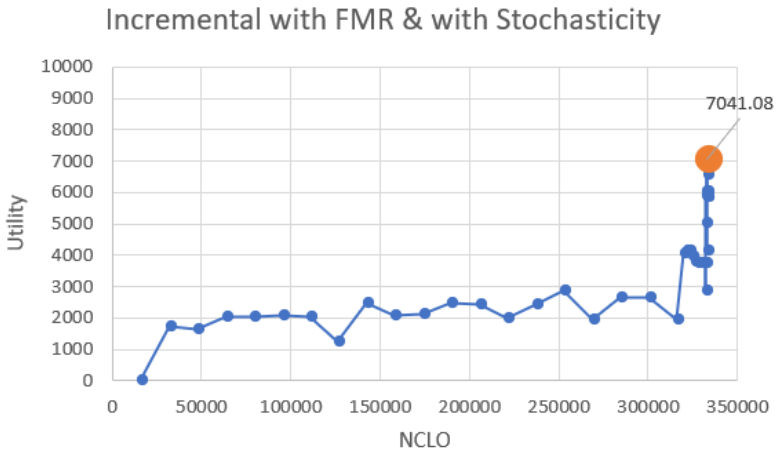
3.7.3 Single Run – Incremental Max-Sum



In the graph above we see a single run of incremental Max-Sum without modifications. As expected we get low utility values but eventually, we arrive at a higher value compared to One-shot with the same settings. That was expected as the incremental Max-Sum operates at a higher resolution. What we witness at the right of the graph is a phenomenon due to the computational imbalance. Skills get depleted and thus NCLOs drop rapidly and so we get many data points packed densely.



In the graph above we have added FMR to the previous run and thus utilities raised higher as expected.



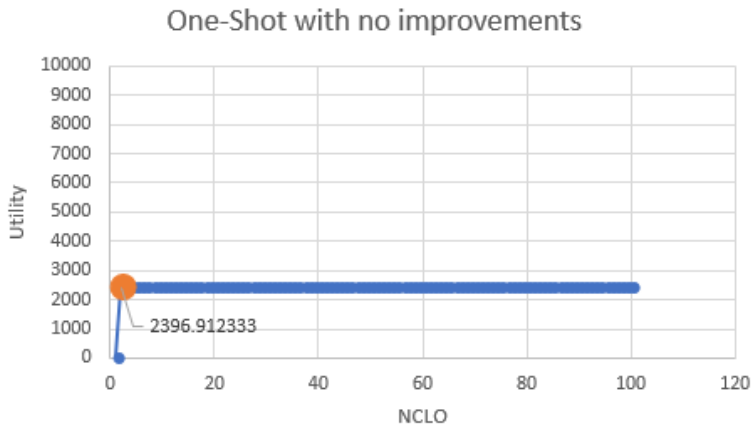
Finally, in the graph above we have added the stochastic element we mentioned before and we see the impact again is noticeable and we get better results.

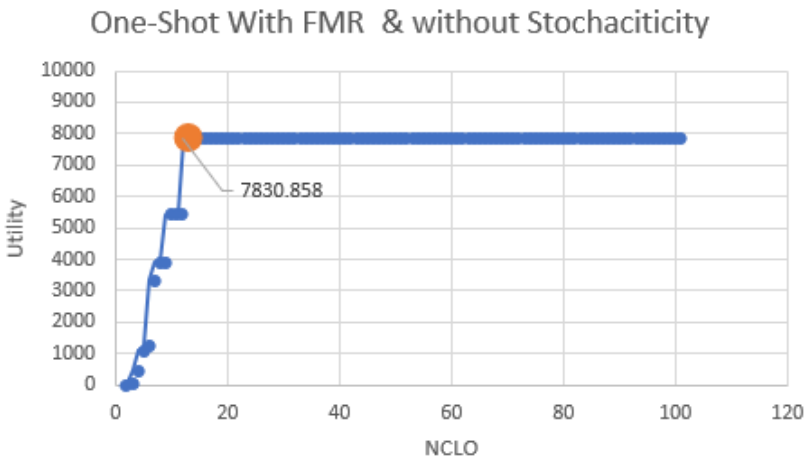
In the final analysis, One-shot got better results in the exact same problem. We speculate that there is a good reason for this result. One-shot has a built-in priority where providers stay and give service to the same requester until all of their skills are depleted. This strategy cuts travel times. That fact, can to some extent, increase the utility gain. On the other hand, what we witnessed with the incremental version is that it has no such priority and therefore agents tend to leave requester they have just visited even though it is a costly decision. Moreover, Max-Sum in general does not have any mechanisms to guarantee cooperation between providers. Thus, One-shot capitalizes on one beneficial strategy while the Incremental version shows no signs of smart decision making. Having said that we also witnessed superiority of Incremental Max-Sum at more plentiful problems, but the painfully long runtimes prevented us from fully investigating how or why the

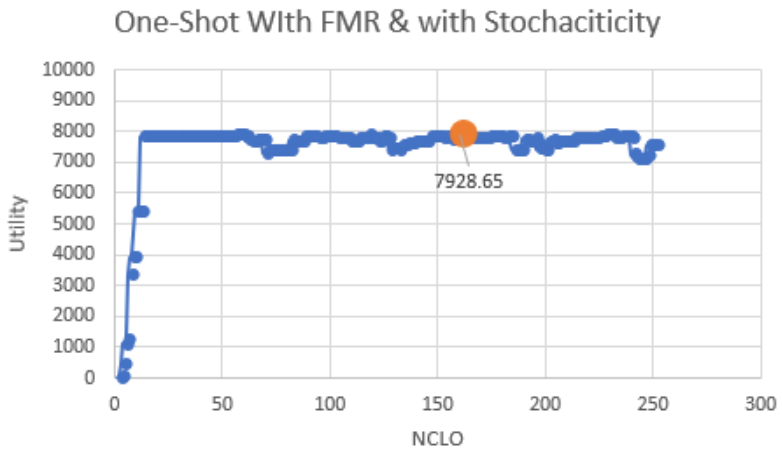
incremental version gets better as the input increases in size. We speculate it has to do with the resolution of decision making. As the problem gets larger the incremental Max-Sum won't overshoot much in the assignments. Meaning the amount of service that benefits no utility at all will be minimized. In the One-shot version though overshooting could become more prominent.

3.7.4 30 Problems – One-Shot Max-Sum

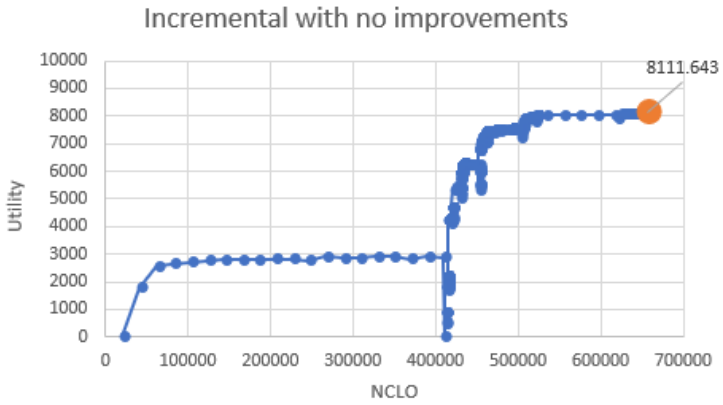
In the following graphs we witness the same behaviour and same conclusions as before, the main difference is that we average the results of 30 problems with the same parameters.

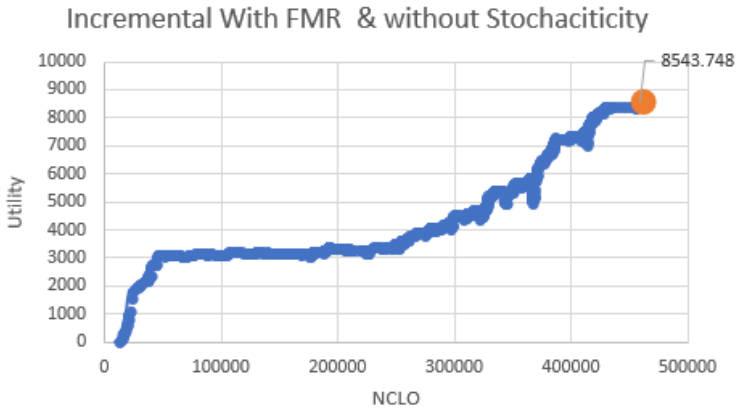


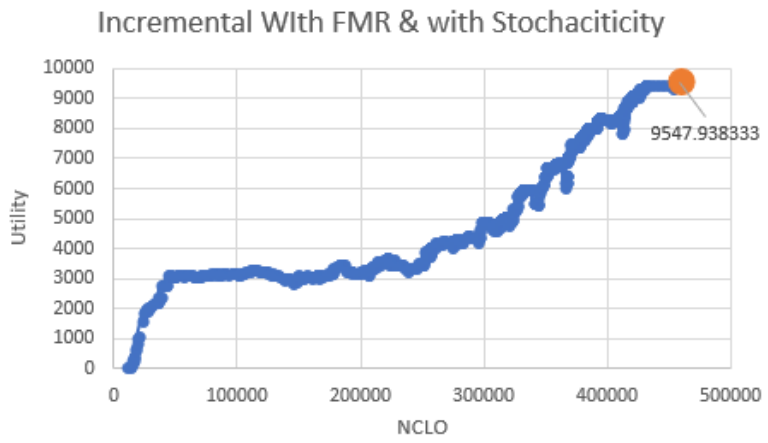




3.7.5 30 Problems – Incremental Max-Sum

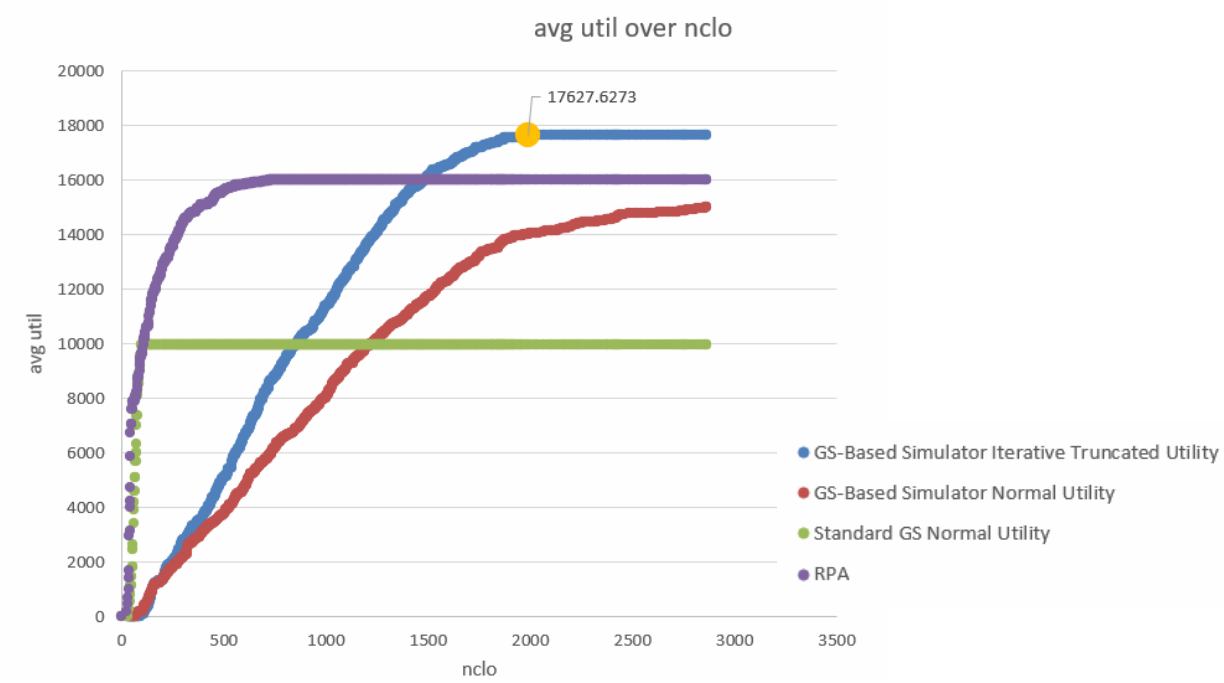


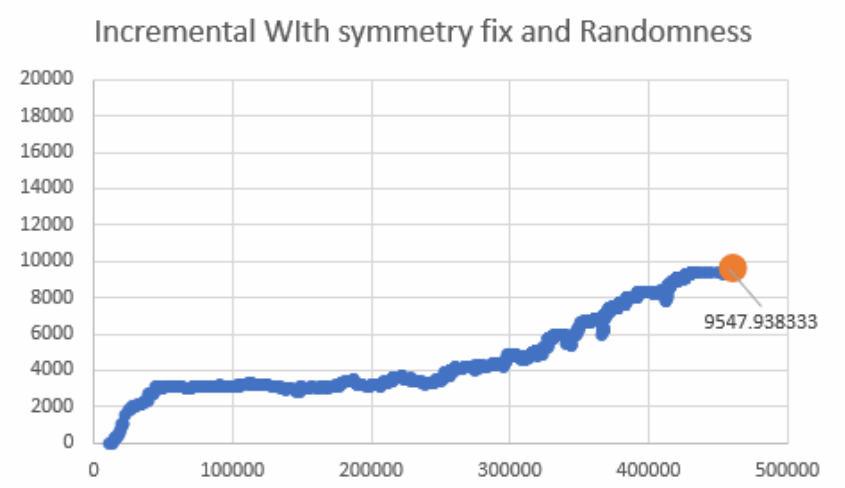




We witness that when averaging 30 problems, the incremental version operates better overall. That means that unlike before, on average most problems generated are effected more by the resolution of choices made and not by the benefit of our one-shot that prioritizes providers staying in their assigned requester until their skills are depleted.

3.7.6 Empirical Comparison of Incremental Max-Sum and known solutions of other algorithms





The overall utility achieved by Max-Sum is lower compared to differing algorithms that solve the same problems.

3.7.7 Conclusions from the experiment

All in all we can say confidently that the Incremental Max-Sum is superior to One-shot in finding better solutions. But as we just saw above, Max-Sum is not the best algorithm for solving service-oriented problems. It is both computationally heavy and it is exploring low quality solutions.

4. Summery

4.1 Summery of project achievements and product

Applications of distributed optimization algorithms in multi agent service-oriented systems is a new scientific frontier. As such, a lot of research is required to sort through the many ideas in that domain. The way to traverse the rough landscape of this newly discovered frontier relies mostly on common sense and intuition. There are many ideas that could get us great results and as many if not more ideas that could fail us.

The problem we had at hand was a problem that could be formulated as a DCOP and it was clear it can be modelled on a bi-partite graph due to the clear distinction between two groups of agents. Common sense and intuition led us to investigate Max-Sum, and algorithm that is fitted for solving DCOPs modelled on bi-partite graphs. Yet the result was sadly the opposite of good. So, unlike the hope or expectation for a perfect fit we arrived at a conclusion Max-Sum is better used elsewhere, not in task allocation problems. That is since there are better approaches to the problem.

It was interesting to learn that unintuitive behaviour Max-Sum would have. We believe Max-Sum still has many things to offer and it can be further researched and explored and we hope to learn more about it.

4.2 On a personal Note

In our final project we have learned a lot. There were a lot of difficulties and hurdles. We had to think for ourselves and come up with creative ideas to solve problems that unexpectedly revealed themselves to us. At times we had to work under strict time limits and we had to deliver something to show for at the end of our deadlines. That type of work process and flow is not untypical in the university, and we expect it is not different everywhere else, meaning at different workspaces in the industry. We are grateful for the opportunity given to us to help research a highly interesting subject in the field of computer science. It assisted our personal growth as students and individuals and expended our minds and thoughts into uncharted territory. It was refreshing, challenging, and a great experience overall.

Finally, we would like to thank our supervisor and guide, Professor Roie Zivan which started raising us under his wing as early as at our second year in the university. Teach us about "Object oriented programming" in a very enthusiastic and interesting way. Later we have met him again at the course "Industrial Internet of things" and finally he became our supervisor for this project we hereby

submit. Without his teachings and care we would not have been able to ever achieve what we have achieved here and ideas would have been left unexplored.

We also would like to the academic staff studying and working under Prof. Zivan. We would like to thank Mrs. Maya Lavie, Dr. Tehila Caspi and Mr. Arseny Pertzovskiy. They helped us and guided us along our journey. Freed time just to sit with us and assist us in so many different ways. Their guidance and good will was extraordinary and much appreciated.

References

[1] Roie Zivan, Steven Okamoto, Hilla Peled. (2014, February 28). Explorative Anytime Local Search for Distributed Constraint Optimization. pp 1-2

[2] Roie Zivan, Liel Cohen, Rotem Galiki. (2019, December 2). Governing Convergence of Max-Sum on DCOPs through damping and splitting, *Elsevier*, pp

[3] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss (2002, January), Understanding Belief Propagation and its Generalizations.

[4] Roie Zivan and Hilla Peled. (2012). Max/Min-sum Distributed Constraint Optimization through Value Propagation on an Alternating DAG. pp 5-6

[5] Harel Yedidsion, Roie Zivan and Alessandro Farinelli. (2018,May). Applying max-sum to teams of mobile sensing agents. pp 2-7

[6] Wooldridge, M. (2009). An introduction to multiagent systems. John Wiley & Sons.

[7] Scerri, P., Farinelli, A., Okamoto, S., & Tambe, M. (2005, July). Allocating tasks in extreme teams. In Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems (pp. 727-734).

[8] Macarthur, K. S., Stranders, R., Ramchurn, S. D., & Jennings, N. R. (2011). A distributed anytime algorithm for dynamic task allocation in multi-agent systems.

[9] Z. Chen, Y. Deng, T. Wu, Z. He (2018) A class of iterative refined max-sum algorithms via non-consecutive value propagation strategies, Auton. Agents Multi-Agent Syst. 32(6) (2018) 822–860.

[10] C. Yanover, T. Meltzer, Y. Weiss. (2006). Linear programming relaxations and belief propagation – an empirical study, J. Mach. Learn. Res. 7 1887–1907.

[11] A. Farinelli, A. Rogers, A. Petcu, N.R. Jennings. (2008). Decentralized coordination of low-power embedded devices using the max-sum algorithm, in: AAMAS. pp. 639–646.

[12] S.M. Aji, R.J. McEliece. (2000). The generalized distributive law, IEEE Trans. Inf. Theory 46(2). pp. 325–343.

[13] R. Zivan, T. Parash, L. Cohen, Y. Naveh. (2020, January 1). Applying Max-sum to asymmetric distributed constraint optimization problems.