Computer Science and Information Technology 4(2): 48-71, 2016 http://www.hrpub.org
DOI: 10.13189/csit.2016.040202

SLUM: Service Layered Utility Maximization Model to Guide Dynamic Composite Webservice Selection in Virtual Organizations

Abiud Wakhanu Mulongo[1,*], Elisha T. Opiyo Omulo[1], Elisha Abade[1], William Okello Odongo[1], Bernard Manderick[2]

[1]School of Computing and Informatics, University of Nairobi, Kenya
[2]Artificial Intelligence Laboratory, Free University of Brussels, Belgium

Abstract Dynamic webservice composition is a promising ICT support service for virtual organizations. However, dynamic webservice composition remains a nondeterministic polynomial (NP) hard problem despite more than 10 years of extensive research, making the applicability of the technique to problems of industrial relevance limited. In [48], we proposed a layered method, *SLUM*, to combat the problem. Analytically, SLUM overcomes the relative weaknesses of two widely used approaches in the literature – the local planning (hereafter L-MIP) strategy and the Mixed Integer Programming (S-MIP) method. Despite the promising benefits of SLUM, it's unknown to what extent and under what circumstances SLUM is better or worse than L-MIP algorithms and S-MIP. The research objective of the study was to investigate the relative performance of SLUM w.r.t S-MIP and L-MIP using two performance criteria: - *solution quality and CPU running time.* Several randomly generated two task workflows of monotonically increasing hardness in the number of webservices per task were used to benchmark SLUM against the other two algorithms. A set of numerical and statistical techniques were used to experimentally compare the solution quality and the running time growth of SLUM against L-MIP and S-MIP. We determined that SLUM generates solutions with an average quality of 93% w.r.t the global optimum. Further, we show that SLUM yields solutions that are 5% more quality than L-MIP. On the other hand, we established that L-MIP outperforms both S-MIP and SLUM by multiple factors in terms of computational efficiency. However, we find that for problem instances with less than 22 webservices per task, S-MIP is about 1.3 times faster than SLUM. Beyond $n=22$, the running time of SLUM t_{eB}, expressed in terms of the running time of S-MIP t_{eA}, is given by $t_{eB}= t_{eA}^{0.78}$. We also establish that SLUM is asymptotically 3.6 times faster than S-MIP on average. We conclude that in order for a virtual enterprise broker to obtain maximum benefit from dynamic service composition, the broker should combine the three techniques in the following manner- (1) for service request without global constraints requirements, L-MIP is the most suitable method to use, (2) Where there is need for global constraints and the number of service providers per task is less than 22, S-MIP is most preferred and (3) in scenarios the number of service providers per task is more than 22 and there is a need to satisfy global constraints, SLUM is superior to both S-MIP and L-MIP.

Keywords Web Service Composition, Virtual Organizations, Hierarchical, Multilayer, Decomposition, Layering, Optimization, Mixed Integer Programming, Service Layered Utility Maximization, Empirical Complexity

1. Introduction

Dynamic webservice composition is an essential ICT support service for virtual organizations as implied in the ICT Infrastructure reference framework for collaborative networked organizations in [42], [64] and the virtual organization reference architecture by Molinah et al [1]. But dynamic webservice composition, remains a NP hard multiple criteria decision making problem [7], [10-12]. despite a decade of research on the topic. This limits its viability for problems of industrial relevance. The two complementary techniques in the literature widely used to combat the problem are (1) local planning strategy, herein dubbed L-MIP. This technique is provably polynomial time but suboptimal in addition to lacking support for global workflow constraints on webservice quality of service, (2) the Mixed Integer Programming method initially formulated by Benatallah et al [9] herein *S-MIP*. S-MIP has the ability to capture global workflow QoS constraints, guarantees

globally optimal solutions but not scalable for workflows having more than 40 webservices per task [9].In [48], we proposed SLUM, a layered method that analytically, overcomes the relative weaknesses of L-MIP and S-MIP. Whereas the relative performance of L-MIP and S-MIP in terms of solution quality and running time has been adequately studied and empirically benchmarked, and consequently the situations under each of them is superior to the other validated and known, the performance of SLUM w.r.t L-MIP and S-MIP has not been empirically validated. Therefore despite the promising benefits of the method as proposed in [48], it remains unknown to what extent and under what circumstances SLUM is better or worse than L-MIP and S-MIP in terms of quality of solutions and computational efficiency. The purpose of this study was to investigate the performance of SLUM experimentally w.r.t S-MIP and L-MIP using two performance criteria: - *solution quality and CPU running time.* The rest of the paper is organized as follows. In section 1.1, we discuss the background and motivation. The general research problem is elaborated in section 1.2. Existing solutions to the problem and gaps are discussed in 1.3. The overall research objective is framed in 1.4 and the specific research questions the paper address are in 1.5. In section 2, related work is presented. Chapter detail the research process and methodology used to answer the research questions. In section 4, the results are presented and discussed. Our contributions are summarized in section 5.

1.1. Background, Context and Motivation

A virtual organization (VO) is a dynamic, temporary, and strategic alliance of many independent and heterogeneous firms (that have their unique core business competencies), that are logically interconnected using Information and Communication Technology networks [42], [1],[2] , [3] [1-3]. The need for business agility, value added services, efficiency in service delivery of value added services and customer centricity are distinguishing survival strategies for global virtual organizations [42], [1], [2], [3] [1-3]. In their ICT-Infrastructure reference framework for collaborative networked organizations, [42] identifies *webservice composition* as an essential ICT service required to facilitate inter enterprise business process integration and coordination towards fulfilling a complex consumer request. A web service is a distributed software component that enables machine to machine interaction over the network using standard network protocols such as the Simple Object Access Protocol and *REST*. In ICT-enabled VOs, webservices are the software components that produce the data required to execute one of the business tasks required to fulfill a particular business process e.g an online purchase order process. Webservice composition on the other hand, is a process that involves the discovery, selection, linking and execution of a set of atomic distributed webservices in a specified logical sequence in order to service complex customer request that none of the services could fulfill singularly [2][3][4]. In the context of VOs, the different distributed webservices are each owned by geographically disperse entities called virtual enterprises (VE), where a VE is formally defined according to Mon et al [1]. Under the VO reference architectures by Mon et al [1], [42], [64], among many other things, the responsibility of implementing webservice composition lies with a business entity called *Virtual Enterprise Broker (*VEB). From the point of view of the service consumer, a VEB is the service provider. By leveraging webservice composition, a VEB is able to define a business process detailing the logical sequence of tasks that should be performed in order respond to some consumer need. The VEB then automates the business process into a computer executable workflow using standards such as the Business Process Execution Language, and business process execution engines such as the Oracle, IBM or Activiti. The workflow is also referred to as an abstract composite service [29]. Once an abstract composite service is defined, a concrete service composition is required to link each of the abstract tasks within the workflow to some concrete executable webservices. The result of a concrete service composition process is a concrete composite service [29]. The execution of an abstract workflow, causes the concrete composite service to be executed to produce the outcome desired by a service requestor. When generation of the concrete composite service is guided by a well-defined abstract workflow, the service composition process is called *workflow flow based webservice composition* [29], [33], [41]. An alternative to workflow based composition is AI based composition. AI approaches aim at fully automated web service composition without human intervention – both the logical sequence of tasks to be performed and the web services to be linked through the sequence are unknown a priori; they have to be established only at runtime automatically based on the knowledge inferred from a service request [33]. However, despite many years of scientific research on the subject, the AI techniques are still far from real and are yet to find their way into industry [7]. Therefore currently, owing to its worldwide adoption and strong industry support, workflow based service composition remains the only viable option for VOs. Our focus in this study therefore is on workflow based webservice composition.

By exploiting worklow based webservice composition, Virtual Enterprise Brokers, can quickly define a new business process, automate the workflow and assemble already existing webservices owned by different virtual enterprises, to execute the workflow to produce a more value added composite service that satisfy a certain market demand. The reuse of already existing business capabilities and already existing software technology components promotes the business agility differentiating factor of VOs and reduces time to market.

However, the degree of business agility of a VEB does not only depend on how fast the VEB is able to generate a composite service from already existing atomic webservices, but also on how well the generated composite service

satisfies specific quality of service (QoS) requirements demanded by different service requestors from time to time. The implication is that the workflow based service composition strategy chosen by the VEB needs to be highly adaptive and sensitive to the webservice QoS requirements of each and every service consumer at all times. If the VEB is able to achieve this requirement, then the *customer centricity* distinguishing element of VOs would be a reality. Dynamic workflow based webservice composition is a promising technology solution to the business challenge. In dynamic workflow based web service composition, the web service that is to execute a workflow task is unknown a priori until the workflow is executed in response to an external service request. In this case, when the workflow is invoked, the set of web services that best answer the demands of the request at a point in time has to be first discovered ,selected from a service repository and invoked in the logical order enforced by the workflow. Dynamic workflow based service composition is contrasted from *static workflow based webservice composition,* in that in the latter, each workflow task is bound to a known web service in advance at design time and the binding can only be altered manually. Static workflow based webservice composition is easier to implement than dynamic webservice composition. Moreover, static webservice composition suffices in business cases where there is no need for customizing service responses in line with consumer specific QoS requirements. Further, if there was negligible variance in the QoS attributes of each component webservice all the time, then the use of static webservice composition would still produce the best known composite webservice, whichever combination of webservices is defined at design time. However, in VOs, customer QoS needs will vary from time to time and from customer to customer. Secondly, in real service environments, the QoS attributes of individual webservices will vary both at design time and at runtime. Considering the two scenarios, static service composition becomes severely limited. With the two main challenges, dynamic webservice composition is more suitable for VEBs.

The successful implementation of dynamic webservice composition would offer the VEBs the following benefits. (1) Improved likelihood of the service consumer obtaining high quality solutions. Even in the event that no suitable solution is found that satisfies the consumer, the user can be provided with the list of feasible solutions and choose whether or not one of them nearly satisfies them, (2) Through re-planning strategies, workflows that are dynamically bound to webservices at runtime are more likely to survive failures through selection of different execution paths hence boosting system reliability and customer experience.

1.2. The Dynamic Workflow Based Webservice Composition Problem

Despite the potential benefits of dynamic webservice composition to VOs especially as highlighted in section 1.1, over the last ten years, dynamic webservice composition remains a nondeterministic polynomial time (NP) hard Multiple-Criteria Decision Making (MCDM) problem [7], [10-12]. This means that in some cases, the time taken to find a solution that meets the requirements of a service consumer may be infinitely large and practically unacceptable to the user. The coupling of a wide range of issues accounts for the difficulty. First, the large number of functionally similar web services within a virtual organization. Discriminating the services against some QoS factors is essential yet nontrivial as shown in [48]. Secondly, the large number of differentiating QoS attributes as evidenced by the papers in [6-9]. This increases the number of optimization constraints over the QoS attributes and makes the optimization process more complex [48]. Third, the volatility of the Service Environment. During the composition process, workflow execution may fail due to various reasons ranging from web services initially available at the start of the workflow being unavailable shortly, unresponsive web services leading to timeouts, server side internal errors etc. These challenges necessitate embedding transaction management and fault handling strategies within the composition process to ensure that workflow execution is sustained in the presence of faults. Fourth, complexity of Workflow Composition Patterns: A business workflow can take on one or a combination of the following patterns: sequential, parallel, OR, XOR etc. The presence of the different patterns contributes varying levels of complexity of issues to be handled. For example, workflows containing parallel patterns present all the challenges inherent in parallel software systems such as consistency and synchronization. We refer the reader to [5], [46] for a comprehensive coverage of the various challenges inherent in dynamic webservice composition. Here, we restrict our study to dynamic composite service selection within a nonvolatile (static) service environment, on sequential workflows, in which the specific problem is:

> *In a virtual organization with a large number of enterprises providing similar services (that are exposed as web services) but differentiated on multiple QoS attributes, auto generate the best composite web service that fulfils a complex service request with multiple QoS constraints*. As stated earlier, this problem remains NP hard.

In the literature, this problem has been tackled using the following general multiple criteria decision making approaches: local planning strategy as described in [9] and [41], hereafter dubbed *L-MIP*, the flat structured global planning techniques in [9] and the layered global planning strategy proposed in [48]. In each of these approaches, the objective is to maximize some utility function over a set of decision variables that are constrained. The utilities are computed using the Simple Additive Weighting [18] model. We elaborate the state of the art in section 1.3.

1.3. The State of the Art Solutions and the Gaps in Existing Literature

The local planning strategy provably finds solutions to the dynamic webservice composition problem in polynomial

time. The studies in [9] and [41] demonstrate this. Thus, local planning would appeal for real time virtual online services. However, local planning does not guarantee global optimality of the solutions produced, and thus on average, the solutions produced by local planning strategies are relatively suboptimal compared to global planning methods [9] ,[41]. The author in [41] experimentally established that local planning methods for the webservice composition problem are worse in quality by 20% to 30% on average relative to the global optimum. In addition to being suboptimal, local planning technique does not take into account global constraints across all the tasks within a workflow, such as the total service execution cost should not exceed a particular budget [9]. The result is that local planning methods might deny a service requestor a chance to express critical global webservice QoS constraints.

Global planning optimization on the other hand overcomes the limitations of local planning models by their ability to express and solve global constraints on workflow tasks. Given sufficient time, global planning is guaranteed to yield an optimal solution. Unfortunately, as demonstrated by Benatallah in [9], global planning methods severely suffer exponential state space explosion for large problems hence obtaining an optimal solution in good time is computationally intractable. This renders global planning methods unsuitable for real-time online applications. For instance the naïve global planning approach uses exhaustive search where it requires comparing generating n^k candidate services and computing utilities for each where n are the number of candidate services per task for k tasks.

An alternative to exhaustive global planning is to apply Mixed Integer Programming, MIP [21] for optimization of composite service selection. MIP is an efficient technique for many optimization problems in which some variables take on integer values while other variables are continuous [22]. The pioneering work in composite service selection based on MIP is by Benatallah et al in [9]. Hence forth, we will call the method in [9] as the standard MIP (*S-MIP).*The authors analytically and empirically show that S-MIP is significantly better than naïve global planning methods. At the same time, the authors in [9] show that although *S-MIP* is generally worse than local optimization, the computational overhead is insignificant provided the problem size is not larger than 40 candidate web services per workflow task. Otherwise the authors note that MIP is still susceptible to the exponential state space explosion in some cases, and thus does not guarantee to find solutions in polynomial time. Therefore S-MIP still limited to small scale service composition problems of 40 webservices per task or less. This limitation is in fact a general inherent limitation of all MIP algorithms beyond webservice composition problem. For example, the authors in 20], note that global planning MIP techniques applied to large scale database query optimization problems suffer exponential state space explosion. However, because of their applicability to a wide range of industrially relevant optimization problems, MIP to date, remains the most preferred approach to solving optimization problems [20].

One would argue that with the current state of the art computing hardware and high performance computing technologies, it should be possible to solve MIP problems very fast. But then it depends how fast. Up to date research shows that even combing the most sophisticated MIP libraries such as the IBM CPLEX with the fastest computing infrastructure would still take a few seconds, through hours to a couple of weeks to solve a given optimization problem, depending on many factors such as the structure and formulation of the problem, the number of constraints, the coefficients of constraint inequalities, the bounds of the on the right hand side of the constraint inequalities [63]. The author in [62], for example proposes an MIP method for an air logistics scheduling problem. The method is tested on top of state of the art IBM CPLEX and hardware. The MIP method takes up to 10 hours on the most difficult problem instances but still it achieves an improvement of over 10 hours on relative to a competing approach. Clearly, 10 hours would be an unacceptable waiting for real time applications but acceptable for the problem the author was tacking, since the specific problem usually takes a couple of weeks to solve. Thus the need for improved MIP methods targeting specific optimization problems such as dynamic webservice composition is glaring.

In [48], we proposed and formulated SLUM: Service Layered Utility Maximization model for the composite webservice selection problem specifically targeting virtual organizations. SLUM extends the S-MIP method in [9] but fundamentally differs from [9] and related methods, in that it follows a two layered approach to the dynamic webservice composition problem, informed by the theory of layering as optimization decomposed as detailed in [24-27]. SLUM is a hybrid of local planning and global planning that trades off global optimality versus speed of execution while taking into account the service requestors global constraints. This achieved as follows. The service composition is partitioned into two sequentially decomposed and connected subproblems called the service consumer utility maximization problem (SCUM) and the service provider utility maximization problem (SPUM). Each of the subproblems is formulated following an approach similar to [9]. The SCUM problem is first solved on a subset of QoS attributes and the services that meet the requirements at this stage are subjected SPUM optimization in a second layer. The model is analytically shown to be more efficient than the flat structured MIP approaches such as [9]. The analysis is based on the theory that even if decomposition is formulated and solved sequentially, the complexity of problems grows superlinearly [22]. Further because of the separation of optimization concerns into service provider and service consumer perspectives, service consumers are shielded from the having to understand the details of low level technical QoS attributes such as reliability, availability and throughput [48]. However, like local planning strategies, SLUM is bound to yield suboptimal solutions relative to S-MIP. The

reason for this is that although SLUM takes into account global constraints across workflow tasks, it only considers a subset of these constraints one at a time.

Whereas the relative performance of L-MIP and S-MIP in terms of solution quality and running time has been adequately studied and empirically benchmarked, and consequently the situations under each of them is superior to the other validated and known, the performance of SLUM w.r.t L-MIP and S-MIP has not been empirically validated. Therefore despite the promising benefits of the method as proposed in [48], it remains unknown to what extent and under what circumstances SLUM is better or worse than L-MIP and S-MIP in terms of time quality of solutions and computational efficiency. The purpose of this study was to investigate the performance of SLUM experimentally w.r.t S-MIP and L-MIP.

1.4. Research Objective

The main objective of this study is to experimentally evaluate the performance of the two layered mixed integer programming model for composite webservice selection, SLUM [48] against SMIP [9] and local planning strategy as described in [9] (L-MIP). Optimization algorithms are mainly evaluated based on the computational effort in terms of CPU running time, and secondly, the quality of the solution Zemel Eitan [51], [47]. In this study, we used the two performance criteria to assess the relative performance SLUM.

1.5. Research Questions

The study aimed to answer the following research questions:

RQ1: How does the solution quality accuracy of SLUM compare with that of S-MIP?

*R**Q2***: How does the solution quality accuracy of SLUM compare with that of L-MIP?

RQ3: How does the CPU running time growth of SLUM compare with that of S-MIP?

RQ4: How does the CPU running time growth of SLUM compare with that of L-MIP?

By answering the above questions, the goal of the study was to determine the conditions under which SLUM is a better or worse strategy than either L-MIP or S-MIP or both.

2. Related Work

Previous work related to our current work can be grouped into two–*monolithic Multiple-Criteria Optimization Strategies* and strategies which employ some decomposition strategy herein referred to *Decomposed Multiple Criteria Optimization Strategies.*

2.1. Monolithic Multiple Criteria Optimization Strategies

The authors in [40] propose a QoS based optimization on Mixed Integer Programming. Although [9] earlier demonstrated that MIP is more efficient than exhaustive search, we saw in section 1 that MIP is still susceptible to exponential explosion. The deficiency in [40] can be said of the MIP techniques in [41], [43] and [44]. Moreover, [43] does not provide any justification for the use of an MIP method based on Taylor expansion.

[42] Combines AI planning with the Simple Additive Weighting method [18] to select the best composite service. This approach is limited to only problems in which constraints can be expressed only as propositional logic. In the web services community, such an approach is restricted to semantic web services only. Additionally, the approach is subject to combinatorial explosion when the number of propositions grow.

A Fuzzy logic based multiple criteria method is presented in [39]. The method involves a user expressing preferences by assigning a Confidence Factor (CF) on the range [0, 1] to Fuzzy rules. A CF denotes the importance of each fuzzy rule. The higher the CF value the more important the Fuzzy rule is relative to another rule. Two shortcomings observed on other approaches are inherent in this approach. Firstly, the fact that a user has to express their preferences by specifying confidence Factors, imply that the approach is limited to technical audience only. Lastly, the complexity of the problem exponentially expands with an increase in the rule base.

2.2. Decomposed Multi-criteria Optimization Strategies

A decomposition method for service selection based on Mixed Integer Programming is presented in [11]. The method works as follows: Global constraints are converted into local constraints. Even though [11] claims reduced MIP model that can be solved in linear time, neither details are on how the decomposition method works nor empirical results supporting the hypothesis are provided.

The two step Mixed Integer Programming algorithm by Alrifai et al [37] is based on decomposition of global constraints into local constraints. After the decomposition, local optimization using local constraints follows. Firstly, each web service QoS attribute value is partitioned into quantized levels for each web service in each service community. The goal is to find the best combination of quantized values that will be used as upper bound constraints within the second step. MIP is applied to find the best combination of values that satisfy the constraints. In the second stage, a local planning selection algorithm is used to select the best web services. The challenge with this approach is that expressing global constraints that will not be violated by local constraints is a challenge. Further, the performance of the model is affected by the number of quantity levels d. The larger the d the less efficient the model becomes and vice versa. The value of d or range of d for which the model can perform better than conventional MIP remains unknown. Thirdly, like all other techniques, the model is still too complex from an end user perspective and

therefore its utility is equally limited to technically sophisticated users.

Similar to [11] and [37], our work in [48], also described in section 2 of this paper, in general follows Mixed Integer programming with a decomposition strategy to optimize composite web service selection. But unlike the rest, the work in [48], combines the well-known MIP global planning technique for web service selection in [9] with the emerging theory of layering as optimization decomposition [24-27]. The distinctive features of this approach are: 1) two distinct objective functions, one addressing the concerns of the end user and the other addressing the concerns of the service provider. Although, in [48], each layer (sub problem) can viewed as a local optimization problem as within the scope of the entire "network", from a web service composition perspective, each of the local sub problem employs a self-contained global planning method using the S-MIP technique in [9], although on a subset of constraints and subset of QoS attributes. This brings the second unique feature: 2) that we are still able to express global constraints, and that these global user and service provider constraints alike, are guaranteed not to be violated, 3) By using layering as a design time architectural style, separating the concerns of the users from those of the provider, we not only achieve more efficiency, but also derive simplicity that shields the average end user from the complexity of technical jargon, and tediousness of having to capture constraints and weight preferences on low level QoS attributes. All the other methods lack this feature.

Although this paper is very closely related to [48], a significant difference exists between the two. In [48], only a qualitative and analytic argument is presented in favor of layering as optimization decomposition based on Mixed Integer Programming. This paper complements [48] by providing detailed quantitative evidence using six different statistical and numerical analyses methods, that reinforce the fact that MIP techniques based on layering as optimization decomposition yield composite services more efficiently compared to monolithic MIP techniques

3. Methodology

According to[50], [53-54], successful empirical evaluation of algorithms entails taking into account the following issues: the fairness of algorithm implementation, the method used to select problem instances also called benchmarks, performance metrics used, experimental protocol and the methodology for data analysis and interpretation. We address each of these issues in the subsections that follow.

3.1. Algorithm Implementation

The SLUM, L-MIP and S-MIP algorithms were all implemented in Java 7.0 using the Java Optimization Modeler (JOM)[1] tool version 1.15 with a GLPK[2] linear programming optimization solver. Each of this algorithms invoked a program function called *getBestCompositeService (int optMode)* where "optMode" denotes optimization type. The enumeration values for the argument is one of L-MIP=0, SLUM=1 and S-MIP=2.

3.2. Performance Metrics

3.2.1. Performance Metrics for Running Time

When measured empirically, the running time of an algorithm can be captured using CPU time or through the use of operations counts. The use of CPU running time often raises validity threats. Ravindra et al [56] for instance, remarks that empirical studies involving CPU running time as a measure of algorithm performance suffer from multiple sources of variability: programming language used, compiler, computer hardware, the approach used to encode the algorithms into computer programs (which depends on the skill of the programmer), problem input size parameter combinations and whether or not at the time of measurements, other users were engaging the computing hardware that is used to execute the experiments. Nevertheless, CPU running time still remains the most widely used method for measuring algorithm running time [47]. On the other hand, operations count entails automatically counting the number of times an algorithm executes major operations (*Ravindra et al, 1996)*. Specific methods of how to implement the operations count method empirically include the *representative (bottleneck) operations counts* method proposed in [56] and the *trend-prof* tool utilizing *execution operation counts* Goldsmith et al [55]. This method ideally targets to curtail the challenges of experimentations based on CPU running time. However, the *execution operation counts* approach is not devoid of limitations. In practice it might be difficult to establish which operations critically affect performance and which ones are insignificant and should therefore be excluded. Moreover, because the counting of such operations is automated through a profiler such as the one in Goldsmith et al [55] the impact of the profiler on the program (system) under test may be another source of variability that would be equally elusive to isolate.

This study adopted the CPU running time since its straight forward and popular. The mentioned validity issues due to CPU running time are addressed in the experiment design approach and in the choice of the analysis methods in section and respectively. The measurement units of time were "seconds". The choice for "second" as a unit of measure is motivated by two reasons: 1) the second is the SI unit of time and 2) MIP algorithms take seconds to weeks to solve given problem instances [63], therefore using the second is

1 www.net2plan.com/jom/
2 https://www.gnu.org/software/glpk/

adequate enough to ensure high precision and resolution of time measurements. For each problem instance, the CPU running time was automatically tracked and recorded on successful termination of the optimization process.

3.2.2. Performance Metrics for Solution Quality

A major goal of optimization algorithms is to generate a high quality solution from a large space of alternative solutions within a reasonable amount of time. A common method to gauge the solution quality of the solution value z produced by algorithm A *(*where A is hypothesized to be suboptimal) on some optimization problem instance p*,* is to compare the value z against a global optimum value z^* output by an algorithm B that is known to produce a globally optimal solution value z^* for every $p \in P$. We know that the S-MIP algorithm has the property that for every $p \in P$, z^* is output. Let $z^B$and z^L be the solution values produced by SLUM and L-MIP on the same problem instance. We are interested in measuring the pairwise solution accuracy between z^B and z^* and z^L and z^*. Two commonly used metrics for comparing a suboptimal or approximate algorithm w.r.t an optimal one, in terms of solution accuracy are *optimality ratio* (OR) and relation solution *quality (RSQ)*. The two metrics have been used previously in studies such as Coffin et al [47] and *Holdger H. Hoos* [50]. To make it more intuitive, we convert RSQ to a percentage by scaling it by 100 as per (1). We denote the RSQ of SLUM w.r.t to SLUM as RSQ_B and RSQ of L-MIP w.r.t to SLUM as RSQ_L so that (2) and (3) follows. The optimality ratio of L-MIP and SLUM are given by (4) and (5) respectively.

$$\text{RSQ} = ([z^* - z)/(z^*)] * 100) \quad (1)$$

$$\text{RSQ}_\text{L} = ([z^* - z^L)/(z^*)] * 100) \quad (2)$$

$$\text{RSQ}_\text{B} = ([z^* - z^B)/(z^*)] * 100) \quad (3)$$

$$\text{OR}_\text{L} = ([z^L)/(z^*)] * 100) \quad (4)$$

$$\text{OR}_\text{B} = ([z^B)/(z^*)] * 100) \quad (5)$$

3.3. Benchmark Selection and Generation of Input Problem Instances

3.3.1. Problem Instance Selection for Comparative Benchmarking of Running Time

When benchmarking algorithms in terms of running time, the three issues to be considered are: 1) instance hardness- the researcher should focus on hard instances, 2) instance size. A range should be provided for scaling studies and, 3) instance type. The researcher ought to provide a variety of problem instances [50], [54]. Problem instance type or variety can be achieved through the use real application instances or ensembles of instances from random distributions Holdger H. Hoos [50].

In regard to problem instance type, this study adopted instance types that are generated from random distributions. Using this approach is more flexible in adjusting experimental factors and less expensive compared to using real webservice instances. In relation to instance hardness and instance size, for CPU runtime, we chose 15 problem instances ranging from the simplest having five candidate webservices per workflow task, to the hardest having 75 candidate webservices per workflow. In between the two were instances whose size was a multiple of five. Therefore problem instances of size *n=5, 10, 15,… 75* were considered. The fifteen problem instances provide an adequate variety, given that in a similar study in [9], four problem instances were used. Similarly, the hardness of the problems ranging from 5 to 75 is sufficient since in related such [9], the problem hardness in terms of number of webservices per task is varied from 10 to 40. However, it's worth noting, that while the instance type (statistical structure) affects the running time of randomized algorithms, it does not affect the running time of exact algorithms. Since all the three algorithms are exact algorithms, our focus was on how variation in the problem instance size affects the running time of each of the three algorithms.

3.3.2. Problem Instance Selection for Comparative Benchmarking of Solution Quality

The same considerations as those stated in section 3.3.2 were taken into account when generating benchmarks for solution quality (SQ) comparison. Unlike in 3.3.2, where the focus is on the hardness, here the focus is how the SQ of SLUM and L-MIP compares on a variety of independent problem instances. Hence for SQ, 40 randomly generated problem instances having *n*=2, 3, 4,..., 41 webservices per task were used. In section 3.5, the choice of 40 problem instances aided in the selection of appropriate statistical tests of significance as explained in section 3.5.

As noted earlier, the S-MIP [9] algorithm guarantees global optimality while local planning has no guarantee for global optimality i.e it may yield suboptimal solutions, as experimentally illustrated in [9]. The SLUM algorithm in [48] is hypothesized not to find globally optimal solutions at the "network level" in some cases since it does not consider all QoS attributes at ago, even though it does guarantee optimality within each layer (since global constraints across workflow tasks within a layer are considered). Thus both L-MIP and SLUM are somewhat approximate optimization algorithms relative to S-MIP. Whether or not SLUM or L-MIP finds a global optimum and if not how close the suboptimal solution is from the global optimum may vary from problem instance to problem instance based on the structure of the problem instance. To illustrate this, we will denote G_i as the input webservice QoS graph to a problem instance $P_{i..}$ G_i contains k webservice QoS matrices where matrix M_1, M_2 .., M_k corresponds to the set of webservices that can execute worklow task 1, task 2, and task k correspondingly. The vectors within each matrix are of a fixed length v where v is the number of QoS attributes. Assume v=7 so the QoS attributes in consideration are the 7 QoS attributes

according to [48]. For ease of readability, we use "{ }" to represent a graph, "[ ]" to represent a matrix within a graph, "⟨ ⟩" to represent a vector within a matrix, "," is used to separate vectors within a matrix, and matrices within a graph. Let G_1 and G_2 be two webservice QoS graph instances defined according to (2) and (3). Examining the two graphs, G_1 seems somewhat systematically and selectively chosen because in each matrix of G_1, one QoS vector (webservice) dominates the other one on all the 7 QoS attributes such that selecting the dominant (best) webservice from each matrix using L-MIP yields a solution global solution. Similarly, selecting the best composite using SLUM will yield network wide global optimum because in layer 1, vector 1 of matrix 1 will be chosen and in layer 2 the same combination will be chosen. Thus if for all problem instances, one vector in each matrix of the input webservice QoS graph dominates the rest on all QoS attributes, it would give a false impression that S-MIP, SLUM and L-MIP all yield globally optimal solutions all the time or all the three at least yield the same cost values (whether optimal or not) for all problem instances.

$$G_{1=}\left\{\begin{bmatrix}\langle 0.99,0.95,1000,5,4,10,20\rangle,\\ \langle 0.91,0.85,800,3,3,20,50\rangle\end{bmatrix},\begin{bmatrix}\langle 0.90,0.88,100,3,4,40,20\rangle,\\ \langle 0.95,0.90,800,5,5\ 15,10\rangle,\\ \langle\ \rangle\end{bmatrix}\right\} \quad (6)$$

$$G_{2=}\left\{\begin{bmatrix}\langle 0.91,0.81,100,5,1,10,200\rangle,\\ \langle 0.97,0.7,80,1,4,40,100\rangle,\end{bmatrix},\begin{bmatrix}\langle 0.90,0.83,100,2,5,40,80\rangle,\\ \langle 0.89,0.96,100,5,4,20,95\rangle,\end{bmatrix}\right\} \quad (7)$$

On the other hand, contrary to G_1, G_2 appears to have a random structure and it's not obvious which webservice within each task is better than the other since some are better than the other on some QoS attributes and worse on other QoS attributes. In this case, it's likely that each of the three algorithms yield different cost values.

Due to the sensitivity of SQ to the structure of problem instances, to ensure plausibility of results on solution quality, for each problem instance P_i, the graph G_i was randomly generated. Both internal and external validity of results as far solution quality is concerned is then ensured because:

- The QoS matrices within each graph instance are randomized eliminating statistical bias. Thus, any differences observed on objective function cost values on S-MIP, L-MIP and SLUM is not due to mere chance of the structure of the problem ,for example systematic dominance of one QoS vector over the rest for each workflow task.
- Each graph instance is independently generated from the other and therefore in differences observed on optimality values across one graph instance is not dependent or related to the other.
- The random graph instances generated have monotonically increasing number of QoS vectors (webservices) per task. From a solution quality perspective, this increases the variety of candidate solutions available.

3.4. Experimental Protocol

In this section, we address several issues that need to be addressed during the actual processing of running the experiments in order to ensure validity of results.

3.4.1. The Protocol in Measuring CPU Running Time

The time taken to find an optimal solution can be affected by the number of constraints. During experimentation, all problem instances as well constraint inequalities remained unaltered. The three algorithms were tested on the same problem instance, one at a time until the entire ensemble of problem instances were exhausted.

Given that different sections of a program can take different times to execute, we only measured the time taken to for each of the algorithms to execute the function *getBestComposite Service (int opt Mode)* in each of the experiments. The Java function *System.CurrentTimeMillis()* was used to compute the time lapse. Secondly, we ensured that each time the experiment was conducted, the CPU and Memory Utilization of the computer used to conduct experiments remained fairly constant. We realized this by having only the default auto start system services and processes running and only our Java system prototype running during each experiment. Thirdly, the same computer was used throughout the experiments. The computer had the following specifications: LENOVO, Windows Professional 64 bit (6.1, build 7601), Intel Pentium CPU B960, 2 CPUs @2.20 GHz, 2GB RAM, CPU Utilization, Physical Memory Utilization State at any one time fluctuated between 88% to 93% at any one time, averaging about 90% (or 1.43 GB of 2GB).

In addition to the above measures, to minimize effects of variance by chance in response time observed on different values of *n,* within and between the two treatments, for every problem instance*, 10* consecutive measurements of time were taken one at time and the arithmetic average value recorded. This was achieved by executing the workflow 10 times repeatedly. 10 was chosen because it's statistically known that 4-10 repeated measurements are sufficient to significantly reduce the random errors observed on measurements due to uncertainties in the measurement environment. Moreover, for every problem instance, the time measurement on each of the algorithms was immediately successive. For example, at *n=10,* 10 successive measurements of time are taken on L-MIP, then 10 successive measurements on SLUM, then 10 successive measurements on S-MIP. The process is then repeated for *n=15, n=20* etc. This protocol differs from the approach where for n=10, 15, 20 .., 10 successive measurements are taken for L-MIP for each *n,* and then the procedure repeated

for SLUM and then S-MIP. The first approach minimizes the time gap between when measurements on the same subject (problem instance) but on a different treatment (algorithm). This is meant to minimize the impact of intervening system state changes with the passage of time. To put this in context, suppose, at n=10, *20* ,30, 40, 50,60,70 ,L-MIP takes 5 seconds ,10 sec, 20 sec, 40 sec, 80 sec, 160, 320 sec to record ten time measures respectively, if the first method is used the 10 measurements at *n=10* on SLUM would be recorded shortly after 5 seconds, while using the second approach, the 10 measurements at n=10 on SLUM would be recorded after 635 seconds or approximately 10 minutes. If the for some reason, the CPU memory utilization bursts within the 10 minutes delay, the pairwise comparison of the system response time when using L-MIP vs when using SLUM at *n=10* would be somewhat biased.

3.4.2. Protocol in Measuring Solution Quality

The solution quality produced by an optimization algorithm is bound to be affected by not only the number of constraints but also the coefficients of the constraint inequalities on the left hand size, the boundary values on the right hand side of the constraint inequalities, and the values within problem instances. We kept these factors invariant and same across the three treatment types. There was no need for taken several repeated measurements of the solution values for each problem instance, since L-MIP, SLUM and S-MIP are exact algorithms that guarantee to ouput same solution for a given problem instance no matter how many times the algorithm is invoked on the same problem instance. This differs from probabilistic optimization algorithms that is bound to yield different solutions at different times, other factors kept constant.

3.5. Performance Data Analysis and Interpretation

The study took a rigorous numerical and statistical approach to the analysis and interpretation of the performance differences of S-MIP, SLUM and L-MIP. Our methodology is mainly informed by the ideas in Coffin et al in [47], Holdger. H. Hoos [50] and in [57], Holdger H. Hoos et al in [58] , Zongxu Mu and Holdger. H. Hoos in [59], Holger H. Hoos and Zongxu Mu in [60] and Goldsmith in [55], [61], [49]. Using the ideas of [47], [49], [50], [55], [57], [60-61], for example, where appropriate, we derive statistical regression models through model fitting, that describe the empirical scaling behaviour of each of the algorithms w.r.t to problem instances of monotonically increasing hardness. If the metric is running time, and where meaningful statistical scaling models relating the growth CPU time and the number of webservices per task are obtained, the performance comparison of the three algorithms is first done using the concept of *empirical complexity (EC)* as described in [47]. EC gives running time performance bounds (empirically) without regard to constant terms, just as is with the case of theoretical algorithm analysis (see section 3.5.2.1). Secondly, motivated by [61], where appropriate and based on the regression models obtained, L-Hospital' Rule is used to determine the average empirical performance of SLUM w.r.t to either S-MIP or L-MIP (see 3.5.2.2 for details). If the pairwise regression models of SLUM and S-MIP or SLUM and L-MIP, satisfy the conditions in 3.5.2.3, the concept of empirical relative complexity [47] is then used to quantify the initial as well the empirical asymptotic performance of SLUM w.r.t S-MIP or SLUM w.r.t L-MIP. In addition to these techniques, we define some descriptive statistical measures of analysis for running time in 3.5.2.5 to augment the analysis. In case no meaningful statistical models for running time were obtained (unlikely to be the case based on the analytic considerations of chapter 1), then the use of sample means or medians coupled with normality or non-parametric tests as described in section 3.5.2.4 were followed.

In regard to solution quality, if meaningful regression models of relative solution quality w.r.t problem instance size *(n)* are derivable, the RSQ models could be used to describe the scaling behaviour of *RSQ w.r.t n* for SLUM and L-MIP. Further, if a scatter plot of RSQ_B vs *n* and RSQ_L vs *n* or *log* RSQ_B vs *n* and log RSQ_L are strongly linear, then the slope test could be used to detect performance solution quality performance differences between L-MIP and SLUM. If no suitable model is derivable or the linearity condition is not satisfied, then either normality tests or nonparametric tests as described in section 3.5.1.2 were used to detect RSQ performance differences between the two algorithms. A similar approach is recommended by Coffin et al [47] and the effectiveness of the approach illustrated by the same author on a wide range of optimization problems in [47].

3.5.1. Analysis of Relative Solution Quality and Optimality Ratio

3.5.1.1. Detecting Solution Quality Performance Difference between L-MIP and SLUM using Slope Test

As said earlier, this test shall only be conducted if linearity exists between both the pairs RSQ_L vs *n* and RSQ_B vs *n* or *ln* RSQ_L vs *n* and *ln* RSQ_B vs *n*. We will generalize and use equation (8) to imply the function describing the linear relation between RSQ_L vs or *ln* RSQ_L vs *n*, and (9) to mean the function describing the linear relation between RSQ_B vs *n* or *ln* RSQ_B vs *n*.

$$y_{in}^1 = \beta_{01} + \beta_{11} n + \epsilon_{ni}\,, \quad (8)$$

$$y_{in}^2 = \beta_{02} + \beta_{12} n + \epsilon_{ni}\,, \quad (9)$$

Where $y_{in}^1 = RSQ_L$ or $y_{in}^1 = ln\ RSQ_L$, $y_{in}^2 = RSQ_B$ or $y_{in}^2 = ln\ RSQ_B$, β_{01} and β_{02} are the intercepts representing heuristic effects [47],, β_{11} and β_{12} are slopes, and ϵ_{ni} are random variations.

We set the null and alternative hypotheses in. (10). If any of the null hypotheses is rejected, then the two algorithms

differ in RSQ performance. Further if H_0 (11) is accepted, then it will concluded that the problem size effect on RSQ_L is the same as the problem size effect on RSQ_B. Similarly if RSQ_B in (10) is accepted, then the heuristic effect on RSQ_B is equal on RSQ in both algorithms.

$$H_{0:}\ \beta_{01} = \beta_{02} \text{ vs } H_{1:}\ \beta_{01} \neq \beta_{02} \quad (10)$$

$$H_{0:}\ \beta_{11} = \beta_{12} \text{vs } H_{1:}\ \beta_{11} \neq \beta_{12} \quad (11)$$

3.5.1.2. Detecting Solution Quality Performance Difference Analysis

To check for performance difference in solution quality between SLUM and L-MIP, we could use the paired Student t-test if the performance differences were normally distributed or non-parametric tests otherwise. To determine normality of the distribution of the performance differences, we used Shapiro Wilk test to test for normality of the performance differences between the pairs. Other tests of normality include Kolmogorov-Smirnov, Lilliefors and Anderson-Darling tests (Razali & Wah, 2011). However, Razal and Wah (2011) established that the Shapiro-Wilk test is more accurate in detecting normality of distribution in data hence the rationale for our choice of the method. With the Shapiro-Wilk test, we computed the statistic W and compared the value of W against the critical value W_c at a significance level of 0.05. The closer the value of W is close to 1 the more likely the data are normally distributed. If W is determined to be more than W_c then it could concluded that there is no reason to believe that the data are not normally distributed, otherwise we could conclude that the differences in RSQ values are not normally distributed. In case, the differences in RSQ values do not follow a normal distribution, we could use either the sign test or the Wilcoxon matched pairs signed rank test. In section 3.3.3, we explained that our optimization problem instances were randomly generated. Holdgers et al [50] if the benchmark optimization problem instances are random in nature, the binomial sign test or Wilcoxon matched pairs signed rank test could be used to detect performance differences (assuming that the test of normality has failed on the data). Since the Wilcoxon tests assume symmetry of the data sample, we would choose the method if a histogram plot of the performances differences was fairly symmetrical in shape. If on the contrary, the histogram was asymmetrical, the sign test that has no symmetry assumptions could be used. In the case of using either of the nonparametric, tests, the null hypothesis is that the median relative solution quality for SLUM and L-MIP is the same at a 95% confidence interval. On the other hand, for the paired student t- test, the null hypothesis is that the mean RSQ_B and the mean RSQ_L are equal.

3.5.2. Analysis of CPU Running time Performance Differences

3.5.2.1. Running time Scaling Analysis using Growth of Functions and Statistical Regression Models

First, we define *empirical complexity (EC)* of an algorithm according to Coffin et al [47] as the function describing the growth of the empirical running time of the algorithm w.r.t to the problem instance size n. Like the theoretical counterpart, the empirical complexity can be regarded as the running time statistical regression model without the constant terms. For example, suppose the statistical model capturing the growth of some algorithm is a polynomial of the form $\beta_1 n^{\beta_2} + \beta_0 n + c$, then as per the definition of Coffin et al [47], the EC of this algorithm is denoted by $O(n^{\beta_2})$.

Here and henceforth, any regression model of the form in (12) will be said to belong to the linear empirical complexity function $O(n)$, while any regression model of the form in (13) will be said to belong to the polynomial empirical complexity function $O(n^{\beta_2})$ and any regression model of the form in (14) will be said to belong to the exponential empirical complexity function $\left(O(e^{o(n)}\right)$. For simplicity, let $C_l = O(n)$, $C_2 = O(n^{\beta_2})$ and $C_3 = \left(O(e^{o(n)}\right)$.

Define $T_{eL}(n)$, $T_{eA}(n)$ and $T_{eB}(n)$ as the parameterized running time empirical functions of the number of candidate webservices per workflow task n, of the local planning optimization strategy as in [9], S-MIP and SLUM respectively. We use $T_e(n)$ to imply $T_{eL}(n)$, or $T_{eA}(n)$ or $T_{eB}(n)$. Let g_l, g_p and g_e respectively, be linear, polynomial and exponential functions of n of the form in (5), (6) and (7) respectively

$$\mu_1 = g_l(n) = \beta n + c \quad (12)$$

$$\mu_2 = g_p(n) = \beta_1 n^{\beta_2} + \beta_0 n + c \quad (13)$$

$$\mu_3 = g_e(n) = \beta_0 e^{\beta_1 n} \quad (14)$$

We would like to establish the running time statistical regression function that significantly describes $T_{eL}(n)$, $T_{eA}(n)$ and $T_{eB}(n)$. To determine whether $T_e(n) = \mu_i$, *i=1, 2 or 3*, we used statistical regression tests where the sample data points were fitted on the model μ_i, i=1, 2, 3, one at a time. The R^2 statistic was used to test goodness of fit of the model μ_i on the data. If the R^2 *<0.8,* we automatically accepted the null hypothesis that μ_i does not fit the data. Otherwise, we performed a further test to check if the percentage of fit is significant. We set $p = 0.05$. If the computed p value is greater than 0.05, we accepted the null hypothesis that the model μ_i, despite having an acceptable goodness of fit, does not significantly fit the data at p = 0.05. Otherwise we accepted the alternative hypothesis that the R^2 value is significant and hence the μ_i model fits the data. The null hypothesis and alternative hypothesis are summarized as below.

$$H_0 : T_e(n) \neq \mu_i \text{ if, } R^2 < 0.8 \text{ or } (R^2 \geq 0.8 \text{ and } p > 0.05) \quad (17)$$

$$H_1 : T_e(n) = \mu_i \text{ if, } R^2 \geq 0.8 \text{ and } p < 0.05 \quad (18)$$

We used the Data Analysis ToolPack and the Real Statistics Resource Pack Microsft Excel 2013 plugins to perform the regression tests

It's possible that for instance $T_{eA}(n)$ is significantly described by more than one kind of regression model e.g $g_p(n)$ and $g_e(n)$ so that (15) and (16) follows.

$$T_{eA}(n) = g_p(n) \quad (15)$$

$$T_{eA}(n) = \mu_3 = g_e(n) \quad (16)$$

According to (15), $T_{eA}(n) \in O(n^{\beta_2}), since\ g_p(n) \in O(n^{\beta_2})$ and $T_{eA}(n) \in (O(e^{o(n)})$ given that $g_e(n) \in (O(e^{o(n)})$ according to (16).

In the case where the running time function is significantly described by two or three regression models, then we concluded that $T_e(n)$ is tightly empirically lower bounded by the smallest empirical complexity class and upper bounded the largest empirical complexity class, noting that $C_l \ll C_2 \ll C_3$. We will use the notation $T_e(n) = \mu_i$ to denote that the $T_e(n)$ *significantly* fits on the model μ_i, $T_e(n) \neq \mu_i$, otherwise where I is a value on the closed interval [1,2,3]. Further, we will use $T_e(n) = \{\mu_2, \mu_3\}$ to imply both the models μ_2 and μ_3 significantly fit the function $T_e(n)$ Thus, $T_e(n) = \{\mu_2, \mu_3\} \rightarrow, T_e(n) \in \{C_2, C_3\}$. If two of the three or all the three algorithms are lower bounded by the same empirical complexity class, then we could conclude that on average, theoretically the algorithms perform the same initially (ignoring the constant terms). If two of the three or all the three algorithms are empirically upper bounded by the same empirical complexity class, then we could conclude that on average, theoretically the algorithms perform the same asymptotically (ignoring the constant terms). If the empirical lower bound of one algorithm *X* is the empirical upper bound of the other algorithm *Y*, then clearly Y is far more efficient than *X*. If two of the three or all the three algorithms share the same lower bound empirical complexity class, as well as the upper bound empirical complexity class, then theoretically, we could conclude that the algorithms have the same runtime performance. As an example, assume $T_{eL}(n) = \{C_1, C_2\}$, $T_{eB}(n) = \{C_2, C_3\}$ and $T_{eA}(n) = \{C_2, C_3\}$. As per our definitions and criteria, we can conclude that L-MIP is far faster than both S-MIP and SLUM, while both SLUM and S-MIP have the same theoretical performance since both of them are have a polynomial lower bound and an exponential upper bound.

3.5.2.1.1. Testing $H_1 : T_e(n) = \mu_1$

This is the most straightforward case. Using the " (linear) regression feature in Data Analysis ToolPack, we provided the $T_e(n)$ values as Y variable values and n values as X variable values. The output was recorded. In addition, a graph of $T_e(n)\ vs\ n$ was drawn and a linear regression trend line added using Microsoft excel native capabilities. The coefficient, intercept and R^2 values obtained using excel trend line feature were then counterchecked against the corresponding values obtained using the ToolPak addin.

3.5.2.1.2. Testing $H_1 : T_e(n) = \mu_2$

Recall that this test entails fitting $T_e(n)$ on the polynomial regression model given in (6) and determining the parameters β_0 β_1, and c for some known β_2. In this work, we set $\beta_2 = k = 2$ thus making the assumption that μ_2 is a quadratic function. This assumption is reasonable in general, since we saw in section 1 that global optimization, algorithms conceptually take time proportional to n^k to generation of candidate composite webservices. Thus by set ting $\beta_2 = 2$, (19) holds.

$$\mu_2 = g_p(n) = \beta_1 n^2 + \beta_0 n + c \quad (19)$$

In (8) we have two independent variables *n (X* variable 1) and n^2 (X variable 2). We computed the range of values of n^2 from known values of *n.* To test that $T_e(n)$ is quadratic, we used multiple linear regression analysis. This was accomplished using Excel Data Analysis ToolPak, where we input the range of $T_e(n)$ values in the Y input range and all the values in the range *X* variable 1 and X variable 2. The output contains among other items the values of the coefficients $\beta_{1\ and}\ \beta_0$ and the intercept C. In addition, a scatter plot of $T_e(n)\ vs\ n$ was done in Excel and polynomial regression fitted on the curve using Excel "Add Trend line" feature. The coefficients and the intercept values and the R^2 values obtained using the scatter plot were compared with those obtained using ToolPak.

3.4.2.1.3. Testing $H_1 : T_e(n) = \mu_3$

As described above, this test involved establishing whether or not $T_e(n)$ fits on some exponential function μ_2 (as in (7)). Since μ_3 is of the form $\beta_0 e^{\beta_1 n}$, if indeed $T_e(n)$ grows exponentially w.r.t *n* then *log* $T_e(n)$ should be linear w.r.t *n*. Following this, we transform (14) to (20) by taking natural logarithms on both sides.

$$\ln T_e(n) = ln\,\beta_0 + \beta_1 n = \beta'_0 + \beta_1 n \quad (20)$$

From (20) we apply linear regression using ToolPak where the *Y* Input range takes on the range of ln $T_e(n)$ and the X input range takes the range of values of *n.* The scatter plot ln $T_e(n)\ vs\ n$ is also drawn and linear trendline obtained in a manner similar to the one explained in the preceding sections. Additionally the graph $T_e(n)\ vs\ n$ (expected to be exponential graphically) is drawn and exponential fitting using the Excel trendline feature done. The β_1 value obtained from ToolPak is directly crosschecked against the β_1 values obtained from the linear regression and exponential regression modes of the scatter plot ln $T_e(n)\ vs\ n$ and $T_e(n)\ vs\ n$ respectively. On the other hand, β'_0 value obtained from the graph ln $T_e(n)\ vs\ n$ is directly counterchecked against the intercept value obtained using ToolPak whereas the inverse of β'_0 or the inverse of the intercept value of the ToolPak output is checked against the β_0 obtained from direct exponential regression fitting of the curve $T_e(n)\ vs\ n$.

3.5.2.2. Running time Expected Relative Speedup Analysis Using Limits of Growth of Functions

Once the performance regression models were

established for each of the algorithms, the analysis that followed in section 3.5.2.1 aimed to characterize the performance of each of the algorithms into empirical complexity classes ignoring the constant terms. The analysis in 3.5.2.1 therefore is tantamount to the theoretical analysis of algorithms. In this section, the algorithms that *theoretically* share the same upper bound empirical complexity class are further analyzed for average (practical) performance using differential calculus and L-Hospital's Rule and limits theory. The rationale for this analysis, is that even if the running time of two algorithms are characterized by the same "worst case" empirical complexity, it's of practical relevance to analyze which one is better than the other on average .Using the example in 3.5.2, even if $T_{eA}(n)$ and $T_{eB}(n)$ have an exponential time upper bound, it is likely that $T_{eB}(n)$ is better than $T_{eA}(n)$, at least informed by the analysis in [48]. In fact, it's easy to quickly tell from the scaling graphs of $T_{eB}(n)$ *vs* n and $T_{eA}(n)$ vs n. Moreover, the coefficients or constant terms obtained through the statistical tests in 3.5.1.1 can hint which of the two algorithms grows faster in running time, even if the two algorithms share the same upper bound empirical complexity class.

We will term the performance efficiency gain (or loss) of SLUM w.r.t either S-MIP or L-MIP as n tends to infinity as *SLUM Expected Speedup (SES).* SES is a function or constants that tells how many times SLUM is faster or slower than S-MIP (if *SES* is computed w.r.t to S-MIP) or faster or slower than L-MIP (if *SES* is computed w.r.t L-MIP). Let SES^g and SES^l represent SES w.r.t S-MIP and SES w.r.t L-MIP. Let $f(n)$ be the empirical regression equation (function) representing the runtime function of SLUM and $g(n)$ be the empirical regression equation (function) representing the runtime growth of either L-MIP or S-MIP. Further, in this analysis, we use $T_e(n)$ to refer to either $T_{eA}(n)$ or $T_{eL}(n)$.

To show that as $n \to \infty$, $T_e(n) \gg T_{eB}(n)$, we need to show that $\frac{T_e(n)}{T_{eB}(n)} \to \infty$, $n \to \infty$. The converse is to show that $\frac{T_{eB}(n)}{T_e(n)} \to 0$, $n \to \infty$. We adopt the former. Applying L-Hospital's Rule, (10) is true.

$$\lim n \to \infty \ \frac{T_e(n)}{T_{eB}(n)} = \lim n \to \infty \ \frac{\partial T_e(n)}{\partial T_{eB}(n)} \tag{21}$$

Mathematically, the SES defined here is given by (22).

$$SES = \lim n \to \infty \ \frac{\partial T_e(n)}{\partial T_{eB}(n)} \tag{22}$$

Thus SES is the slope of the function $T_e(n)$ w.r.t $T_{eB}(n)$ for large enough n. Two outcomes are possible. The first case is that SES is a constant (real number). The second case is that SES is a function of n. In the first case, to show that $T_{eB}(n)$ is more efficient than $T_e(n)$, it suffices to show that $SES > 1$, otherwise for the latter case, we have to show that the function $SES \to \infty, n \to \infty$. Consequently, we make the null and alternative hypothesis below.

$$H_0 : T_e(n) = \ll T_{eB}(n) \tag{23}$$

$$H_1 : T_e(n) \gg T_{eB}(n) \tag{24}$$

H_0 will be accepted if $SES \leq 1 \ or \ SES(n) \to 0, n \to \infty$ .Otherwise H_1

In the case where SES is a function of n, it's essential to determine the value of n for which the value of the slope > 1. We will call the value of n at which the expected speedup is more than 1 as the *expected critical point* and denote it by n_{CE} . The larger the n_{CE} the more remote the chances are that a small scale virtual enterprise broker will benefit from the efficiency of our method as opposed to an alternative technique. The significance of this is so that a virtual enterprise broker, for instance can determine how many virtual enterprise service providers per workflow task the broker needs in order to benefit from using our approach. Because we do not foresee a situation where SLUM is faster than L-MIP, the analysis and determination of n_{CE} will only be w.r.t S-MIP.

3.5.2.3. Runtime Performance Correlation based on Empirical Relative Complexity and Empirical Relative Complexity Coefficient

The analysis in section 3.5.2.1 concerned characterizing the empirical complexity of L-MIP, SLUM and S-MIP with an aim to determining and comparing their theoretical limits. In 3.5.2.2, the analysis targeted (practical –all terms in the regression equation considered) average performance of SLUM w.r.t S-MIP. In this section, the analysis aims to compare the initial practical and asymptotic practical performance of $T_{eB}(n)$ against $T_e(n)$, where $T_e(n)$ could be $T_{eA}(n)$ or $T_{eL}(n)$. The analysis is carried out using the method by Coffin et al [47]. If the regression model obtained by either plotting $T_{eB}(n)$ vs $T_e(n)$ is linear or by plotting $ln\ T_{eB}(n)$ vs $ln\ T_e(n)$ is linear, then the resultant regression model of the form in equation (25) can be used to tell whether $T_{eB}(n)$ is better or worse than $T_e(n)$ initially and by how much, and also show whether $T_{eB}(n)$ is better or worse than $T_e(n)$ asymptotically and by how much. This can be easily being checked graphically. However, graphs wouldn't quantify the magnitude of relative differences initially and asymptotically. Also, the methods in preceding two subsections cannot reveal these levels of detail.

Picking from 3.5.1, and using the example in from table, we saw that $T_{eA}(n)\ \left(O(e^{o(n)}\right)$ and $T_{eB}\left(n\left(O(e^{o(n)}\right)\right)$. By plotting a graph of ln T_{eB} *vs* $ln\ T_{eA}(n)$, a linear regression of the form in (25) holds. Note, $t_{eA} = T_{eA}(n)$. From (25) we obtain (26).

$$ln\ T_{eB}(n) = ln\ \beta_0 + \beta_1\ ln\ t_{eC} + € \tag{25}$$

$$T'_{eB}(n) = \beta_0\ {t_{ec}}^{\beta_1} \tag{26}$$

Thus given the running time algorithm A (S-MIP), running time of algorithm B (SLUM) in terms of t_e can be estimated using (26). Coffin et al [47] refers to the function $O({t_e}^{\beta_1})$ as the empirical relative complexity of algorithm

B relative to algorithm *C* and the parameter β_1 as the empirical relative complexity coefficient of algorithm *B* w.r.t algorithm *C*, where algorithm C is either L-MIP (algorithm L) or S-MIP (algorithm A). When $\beta_1 < 1$, then empirically, *B* is asymptotically much faster than *C* [47]. Otherwise when $\beta_1 > 1$, then , *B* is asymptotically much slower than C [47]. As $n \to \infty, T'_{eB}(n) \approx t_{eA}{}^{\beta_1}$ [47]. On the other hand, when $\beta_0 > 1$, it means that algorithm C is faster than B for small enough *n*, while when $\beta_0 < 1$, it means that algorithm C is slower than algorithm B for small enough *n* [47]. We made two sets of hypotheses, one on the parameter β_0 and the other on β_1. The hypotheses are captured in (27), (28), (29) and (30). The null hypothesis in (27) below claims that the initial performance of both algorithms is equal while the corresponding alternative hypothesis in (28) claims that the initial performance is not the same. H_0 in (29) states that the asymptotic performance of the two algorithms is the same while the alternative hypothesis H_1

$$H_0 : \beta_0 = 1 \quad (27)$$

$$H_1 : \beta_0 \neq 1 \quad (28)$$

$$H_0 : \beta_1 = 1 \quad (29)$$

$$H_1 : \beta_1 \neq 1 \quad (30)$$

3.5.2.4. Performance Difference Detection using Paired Student Test, Sign Test or Wilcoxon Signed-rank Test

As explained earlier on, these tests shall be applied if the performance comparison between SLUM vs S-MIP or SLUM vs L-MIP using the method in 3.5.2.3 is not feasible. Coffin et al [47] notes that CPU running times exhibit increasing non constant variance so that any attempt to use tests with normality assumptions may not yield plausible results. To confirm this, we used Shapiro Wilk test to test for normality of the performance differences between the pairs. If the data were normally distributed, we use the paired student t-test, otherwise we use either the signed test or the Wilcoxon matched paired test.

3.5.2.5. Runtime Performance Analysis using Sample Instantaneous Speedup and Sample Mean Speedup

All the preceding methods of analysis are based on inferential statistics. Inferential statistics provide more rigorous tools (than descriptive statistics) of estimating population parameters based on sample data Howel D.C [52]. Nevertheless, descriptive statistics can be useful tools in summarizing sample data Howel D.C [52] .We define two descriptive statistics: SLUM *Sample Instantaneous Speedup (SSIS) and SLUM Sample Mean Speedup* (SSMS). SSIS and SSMS w.r.t to S-MIP and w.r.t to L-MIP are denoted as $SSIS_g$, $SSMS_g$, $SSIS_l$ and $SSMS_l$, and defined according to (31) , (32), (33) and (34) respectively.

$$SSIS_g = (T_{eL}(n))/(T_{eB}(n)) \quad (31)$$

$$SSMS_g = \frac{(\sum_1^N SSIS_g)}{N} \quad (32)$$

$$SSIS_l = (T_{eL}(n))/(T_{eL}(n)) \quad (33)$$

$$SSMS_l = (\sum_1^N SSIS_l)/N \quad (34)$$

Where *N* is the sample size (number of problem instances).

Thus, SSIS is the speedup of SLUM for a specified problem instance of size *n*. $SSIS < 1$ means that SLUM is slower than the alternative algorithm for some specific value of *n*. $SSIS = 1$, means that SLUM has equal efficiency with the alternative algorithm for some *n* while SLUM is faster than the alternative strategy for some *n* when $SSIS > 1$. $SSMS$ has a similar interpretation, although over the set of all *N* sample problem instances. Similar to n_{CE} (see section 3.5.2.3) will define n_{CS} as the value of *n* beyond which $SSIS > 1, for\ all\ n > n_{CS}$. Thus $for\ all\ n < n_{CS}$, $SSIS \leq 1$. Therefore, even without performing the detailed regression analysis, $SSIS$, . $SSMS$ and n_{CS} can pre-empt some interesting performance behaviour of SLUM. Because we do not foresee a situation where SLUM is faster than L-MIP, we only determine n_{CS} will only be w.r.t S-MIP

4. Results and Discussions

4.1. Measuring Solution Quality

In table 1 above, the optimization solution values for SLUM, L-MIP and S-MIP respectively are given in the columns labelled Z^B, Z^L and Z^* for problem instances of varying size. From the results, we observe that for every problem instance, S-MIP solution value Z^* is the highest value of the three algorithms, implying that S-MIP yields more quality solutions than both SLUM and L-MIP. This result confirms our considerations in section 3.2.2. Thus the result allowed us to compute RSQ_B, RSQ_L, OR_B and OR_L as defined in equations (2), (3), (4) and (5) respectively. In table 1, a RSQ value of 100% denotes that no solution was found (or 100% error rate). For example, we see that for the problem instances with *n=13* and those with *n=14, L-MIP* failed to find a solution where SLUM and S-MIP did. For fair analysis and comparison, we excluded results that contained RSQ=100%. By computing the mean optimality ratio from the data provided, we determine that the mean optimality ratio of SLUM ≈93% , implying an average error rate (RSQ) of ≈7%. On the other hand, the mean OR_L value of L-MIP is ≈88% or $RSQ_L \approx$12%. These simple descriptive statistics suggest that SLUM generally generates more optimal solutions on average than L-MIP by approximately 5%. We shortly validate this claim using one of the tests described in section 3.5.1. Despite the fact that SLUM seems to have a larger mean *OR or smaller RSQ,* there are some cases where L-MIP yields more quality solutions, for example, in table 1 L-MIP is outperforms SLUM when *n*=16, 27, 39 etc. However, as visualized in figure 1 and figure 2, SLUM generally has more quality solutions than L-MIP. With reference to the problem instances with n=*3, n=9, n=12*, it can also be seen that both SLUM and SLUM are able to obtain a globally optimal solutions in some cases.

Table 1. Solution Quality Results on the 40 Problem Instances

N	Z^B	Z^L	Z^*	RSQ_B	RSQ_L	OR_B	OR_L
2	0.69	0.58	0.69	0.00	16.67	100.00	83.33
3	0.61	0.73	0.73	16.44	0.15	83.56	100
4	0.68	0.50	0.68	0.00	26.50	100.00	73.50
5	0.59	0.56	0.75	21.33	25.11	78.67	74.89
6	0.70	0.65	0.73	4.11	10.62	95.89	89.38
7	0.77	0.60	0.77	0.00	22.22	100.00	77.78
8	0.82	0.80	0.83	1.20	3.26	98.80	96.74
9	0.72	0.61	0.72	0.00	14.74	100.00	85.26
10	0.75	0.65	0.79	5.06	17.92	94.94	82.08
11	0.70	0.58	0.72	2.78	19.22	97.22	80.78
12	0.70	0.61	0.7	0.00	13.28	100.00	86.72
13	0.67	0.00	0.7	4.29	100.00	95.71	0.00
14	0.63	0.00	0.69	8.70	100.00	91.30	0.00
15	0.60	0.68	0.68	11.76	0.60	88.24	99.40
16	0.59	0.64	0.75	21.33	15.04	78.67	84.96
17	0.72	0.65	0.75	4.00	13.33	96.00	86.67
18	0.67	0.69	0.75	10.67	7.35	89.33	92.65
19	0.65	0.64	0.72	9.72	10.63	90.28	89.38
20	0.71	0.56	0.77	7.79	27.65	92.21	72.35
21	0.66	0.55	0.74	10.81	25.51	89.19	74.49
26	0.76	0.78	0.69	2.56	11.45	97.44	88.55
27	0.74	0.79	0.79	6.33	0.31	93.67	99.69
28	0.65	0.77	0.60	15.58	21.79	84.42	78.21
29	0.73	0.77	0.74	5.19	4.23	94.81	95.77
30	0.74	0.78	0.68	5.13	13.43	94.87	86.57
31	0.75	0.79	0.77	5.06	2.53	94.94	97.47
32	0.66	0.8	0.74	17.50	6.99	82.50	93.01
33	0.70	0.78	0.67	10.26	13.93	89.74	86.07
34	0.69	0.83	0.71	16.87	14.41	83.13	85.59
35	0.79	0.79	0.60	0.00	24.29	100.00	75.71
36	0.68	0.74	0.62	8.11	15.72	91.89	84.28
37	0.72	0.79	0.72	8.86	8.23	91.14	91.77
38	0.80	0.82	0.74	2.44	9.51	97.56	90.49
39	0.62	0.79	0.66	21.52	17.04	78.48	82.96
40	0.66	0.73	0.70	9.59	4.30	90.41	95.70
41	0.77	0.8	0.63	3.75	20.77	96.25	79.23

Our inferential statistical analysis followed the procedure established in section 3.5.1 to compare the RSQ_B, *vs* RSQ_L, and therefore by implication OR_B *vs* OR_L. Figure 1 is a scatter plot of *RSQ vs n.* An alternative representation of the results is captured by the bar graph in figure 2. The curves RSQ_B *vs n* and RSQ_L *vs n* both hint the following: the variation of *RSQ vs n* is nonlinear, and the performance differences between RSQ_B and RSQ_B do not seem to be constantly increasing with *n.* Thus, the slope test based on linear regression as described in 3.5.1.1 was not found to be an appropriate comparison technique. The alternative approach detailed in section 3.5.1.2 was used instead. Since

our sample size *N=38* and solution quality differences do not exhibit heteroskedasticity, under the assumptions of the central limit theorem, we assume that the *RSQ or OR* are normally distributed. We carried out a Shapiro Wilk test on the 38 sample performance differences, to verify our normality assumption. We obtained *W=0.93* against the critical value W_c=0.938, at a significance of level of 0.05 and 38 degrees of freedom. The result confirms our assumption of normality. Therefore, the paired Student t-test was used to verify the significance of the 5% mean difference in solution quality between SLUM and L-MIP. The null hypothesis is rejected if either t-stat< -t critical two tail or t-stat> t-critical two tail. The results of the significance test are presented in table 2. In table 3 below, the *t-stat = -3.182* and *-t-critical two tail = -2.03*. Since *-3.182 <-2.03*, we reject the null hypothesis and conclude that there are more than 95% chances that SLUM yields solutions with better quality by 5%.

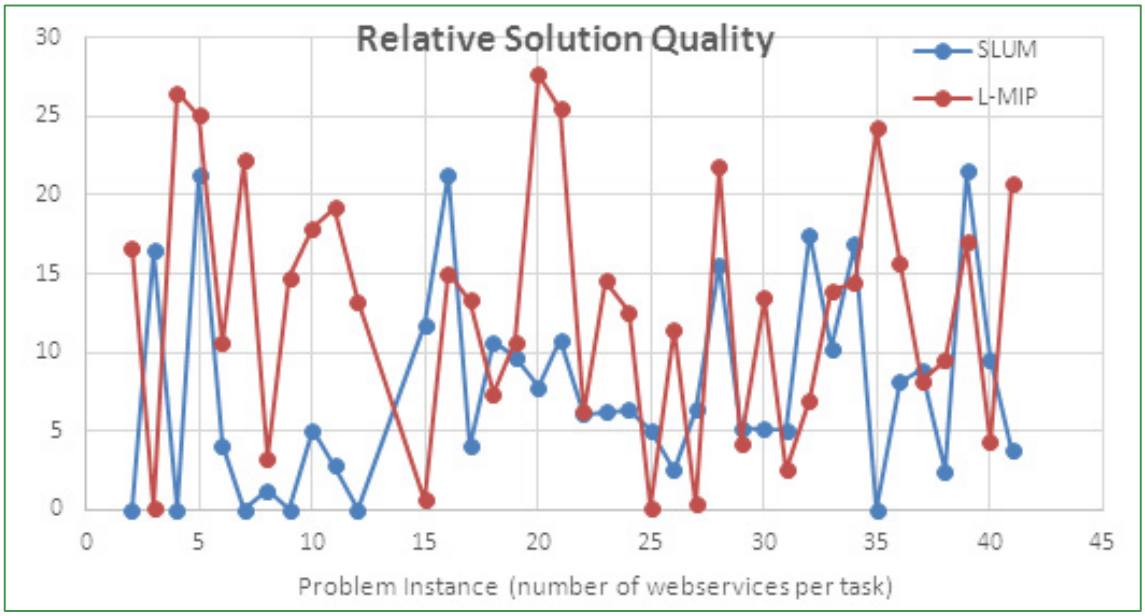

Figure 1. Relative Solution Quality vs Number of Webservices per Workflow task

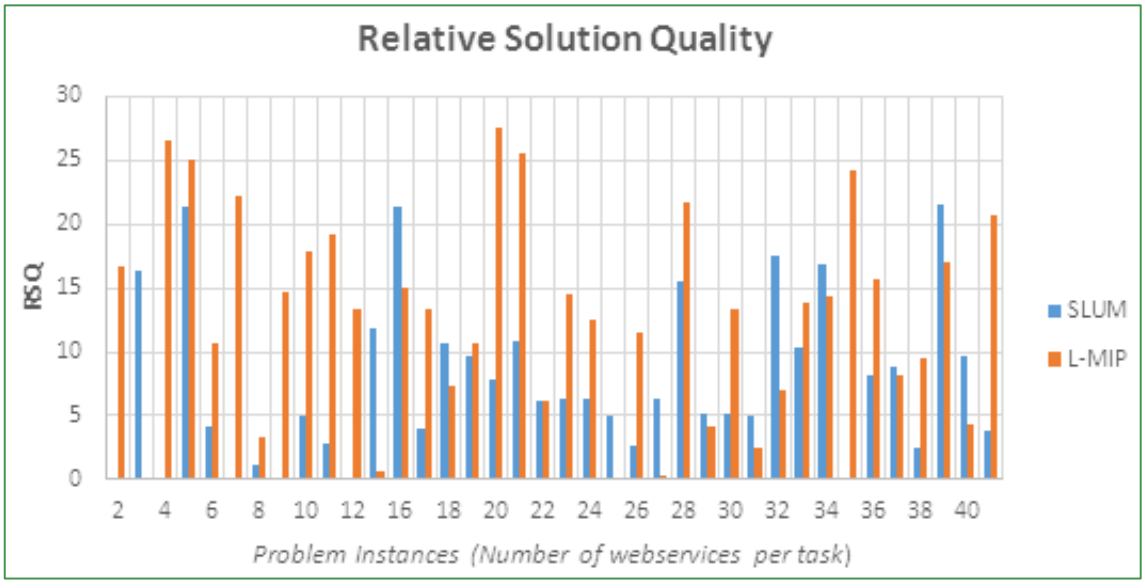

Figure 2. Relative Solution Quality vs Number of Webservices per Workflow task

Table 2. Paired Student T- Test Results On Slum And Limp Relative Solutoion Quality

	SLUM	*L-MIP*
Mean	7.618157895	12.95210526
Variance	40.33246949	62.92169815
Observations	38	38
Pearson Correlation	-0.034624739	
Hypothesized Mean Difference	0	
df	37	
t Stat	-3.182521967	
P(T<=t) one-tail	0.00147773	
t Critical one-tail	1.68709362	
P(T<=t) two-tail	0.00295546	
t Critical two-tail	2.026192463	

4.2. Measuring CPU Running Time

4.2.1. Descriptive Statistical Analysis

It can be observed that for each of the three algorithms, the time taken to find a solution is consistently increasing w.r.t *n*; figure 3 confirms this observation. Moreover, the running time of SLUM, S-MIP and L-MIP nearly the same at *n=5*. Beyond *n=5,* the running time of L-MIP is persistently lower than that of SLUM and S-MIP. In fact, both SLUM and S-MIP running time grows more than double compared to L-MIP for every increment in *n*. For example, when *n*=10, t_{eA} = 1.3, t_{eB} = 1.3 and t_{El}=0.68. Thus, at *n*=10, both SLUM and S-MIP are more than two times slower than L-MIP. When *n* is doubled from 10 to 20, it's possible to see that both SLUM and S-MIP are about 4 times slower than L-MIP. Table 3 also shows that SLUM is generally slower than S-MIP for all *n<40* since we can see that for *n<40*, the SIS<1. On the other hand, SLUM is consistently faster than S-MIP for *n>40* since SIS>1 for all *n>40*.

Table 3. Results on CPU Running Time of Slum, S-MIP and L-MIP

N	t_{eB}(s)	t_{eA}(s)	t_{eL}(s)	ln (t_{eA})	ln (t_{eB})	ln (teL)	SIS_n
5	0.65	0.59	0.55	-0.53	-0.43	0.60	0.91
10	1.3	1.3	0.68	0.26	0.26	0.39	1
15	1.96	1.96	0.8	0.67	0.67	0.22	1
20	4.2	4	0.66	1.39	1.44	0.42	0.95
25	4.6	4.2	0.8	1.44	1.53	0.22	0.91
30	6.9	6.35	0.88	1.85	1.93	0.13	0.92
35	9.9	9.6	0.88	2.26	2.29	0.13	0.97
40	13.3	14.5	1	2.67	2.58	0.00	1.09
45	15.8	22	1	3.09	2.76	0.00	1.39
50	19.3	32.5	1.3	3.48	2.96	0.26	1.68
55	22.5	40.3	1.4	3.70	3.11	0.34	1.79
60	30	62	1.5	4.12	3.40	0.41	2.07
65	37	87	1.66	4.46	3.61	0.51	2.35
70	47	118	1.72	4.77	3.85	0.54	2.51
75	60	155	1.9	5.04	4.09	0.64	2.58

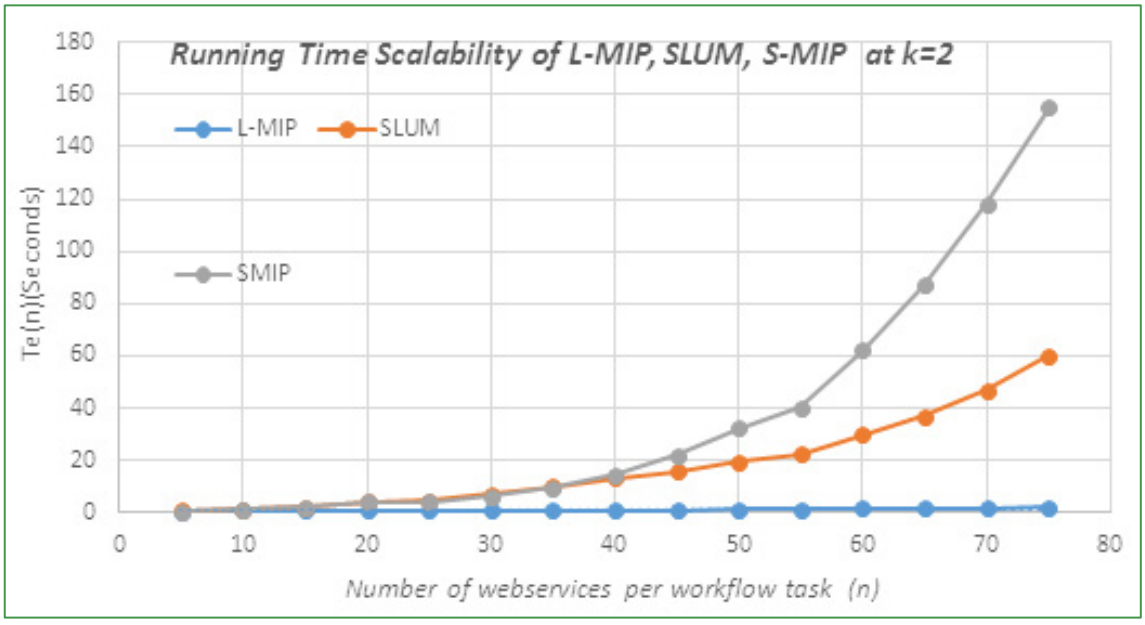

Figure 3. Empirical Running Growth of L-MIP, SLUM and S-MIP

Therefore from table 5 and figure 5, we have that $n_{cs} = 40$.

4.2.2. Measuring Scalability and Empirical Complexity Analysis of Running time Growth

Following the methodology in section 3.5.2.1, table 4 captures the key statistics obtained after fitting the data in table 4 on the linear, polynomial and exponential regression models. Figure 1 depicts the fitting of the running time data for the three algorithms on an exponential regression model while figure 2 and figure 3 show the results of linear and polynomial regression fitting respectively. The fitting was done at a significance level of 0.05. For linear regression p_1 and p_i respectively denote the computed p value for the slope of the model and the computed p value for the intercept. For polynomial regression, p_1 is the p value for the coefficient of *n*, p_2 is the *p* value for the highest order term (n^2) and p_i is the *p* value for the constant term (intercept). For exponential fitting, p_n is the *p* value obtained on the independent variable *n* and p_i is the *p* value for the intercept. As per the analysis method in 3.4.2.1, the results table 5, and in figures 2, 3 and 4, indicate that t_{eL} has a strong and significant polynomial as well as linear growth, even though, L-MIP has a stronger polynomial growth than linear growth.

At the same time, the results show that L-MIP running time is not exponential in growth. SLUM has a weak but significant linear growth, a very strong and significant polynomial growth. S-MIP on the other hand has no linear running time growth but instead has a near perfect exponential growth and a very strong polynomial time growth. Although, SLUM exhibits strong exponential growth as per the R^2 value, the p value obtained on the intercept is larger than 0.05, even though the p value obtained on the slope is far smaller than 0.05. Further, the R^2 of exponential fitting of SLUM is smaller than the R^2 of the S-MIP exponential fitting, suggesting that SLUM has a weaker exponential growth compared to S-MIP. The exponential, linear and polynomial equations representing the growth of CPU running time against SLUM, S-MIP and L-MIP are given in figures 4, 5 and 6.

As per the criteria in 3.5.2.1, we can conclude that the running time growth of L-MIP, $T_{eL}(n)$ is empirically bounded between $O(n)$ and $O(n^k)$ i.e $O(n) \leq T_{eL}(n) \leq O(n^k)$, while $T_{eB}(n)$ *and* $T_{eA}(n)$ are both empirically bounded between polynomial and exponential empirical complexity classes, so that e $O(n^k) \leq T_{eA}(n) \leq O(e^{o(n)})$ and $O(n^k) \leq T_{eB}(n) \leq O(e^{o(n)})$. On the basis of empirical complexity, we therefore conclude that L-MIP guarantees solutions within polynomial time, while both SLUM and S-MIP do not guarantee solutions in polynomial time since they have a polynomial empirical lower bound but exponential empirical upper bound. Thus, L-MIP is far more efficient than both SLUM and S-MIP. Even though, SLUM and S-MIP could be theoretically the same in running time performance, the practical performance differences are intuitive from the table 2 and figure 3, 4, 5 and 6. For instance, from table 2, we determine that the mean sample speedup is ≈1.4, meaning as far as the sample is concerned, SLUM is 1.4 times more efficient than S-MIP on average.

Table 4. Results on Statistical Regression Analysis of Running Time Growth

	Regression Model Type									
	Linear Regression			Polynomial Regression				Exponential (log regression)		
Algorithm	R^2	p_1	p_i	R^2	p_1	p_2	p_i	R^2	p_n	p_i
L-MIP	0.93	$1.95*10^{-5}$	0.0004	0.978	0.86	0.000127	$4.5*10^{-6}$	0.23	0.08	0.35
S-LUM	0.868	$4.3*10^{-7}$	0.007	0.987	0.004	0.000001	0.024	0.97	$1.5*10^{-10}$	0.38
S-MIP	0.78	$2.5*10^{-5}$	0.007	0.98	$6.6*10^{-5}$	$3.0*10^{-7}$	0.0025	0.995	$4.0*10^{-15}$	0.00004

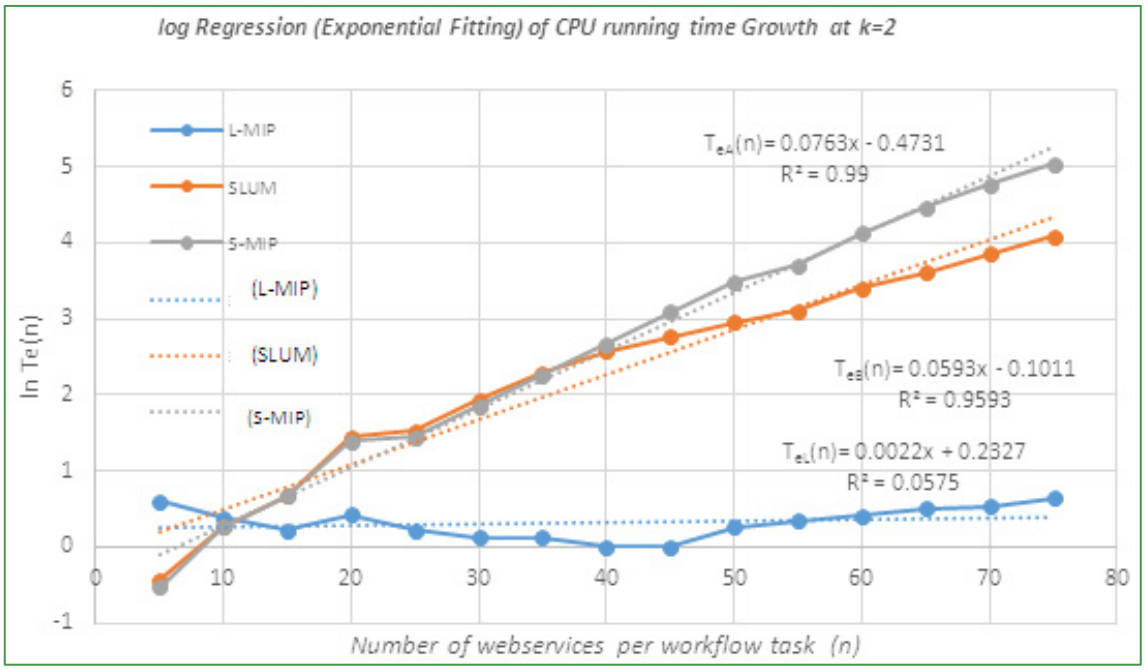

Figure 4. Log $T_e(n)$ vs n. $T_e(n)$ stands for $T_{eA}(n)$, $T_{eL}(n)$, and $T_{eB}(n)$,

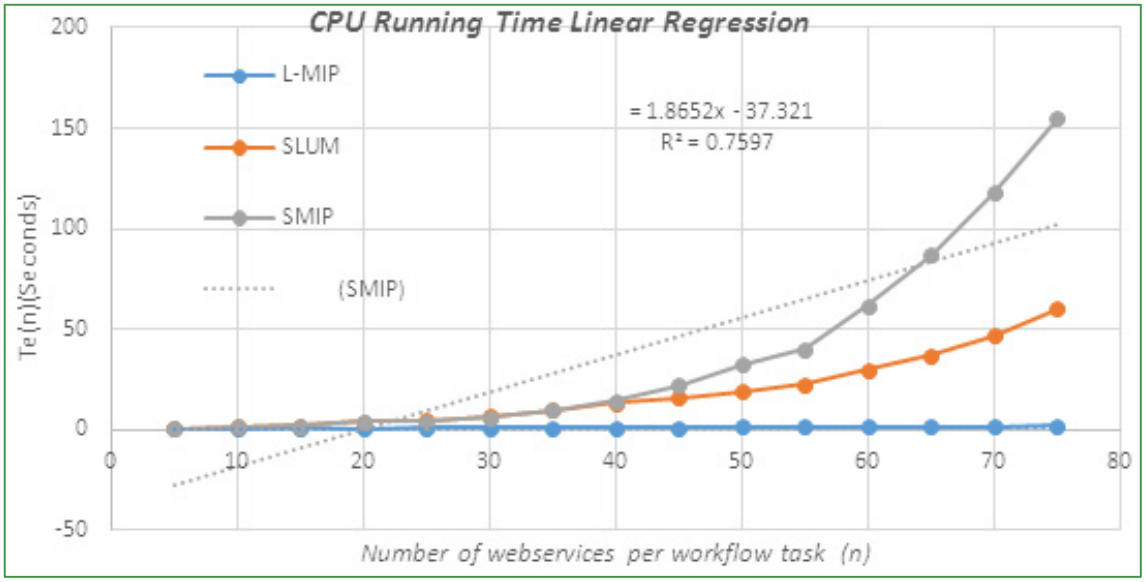

Figure 5. Linear Regression: $T_e(n)$ vs n. $T_e(n)$ stands for $T_{eA}(n)$, $T_{eL}(n)$, and $T_{eB}(n)$,

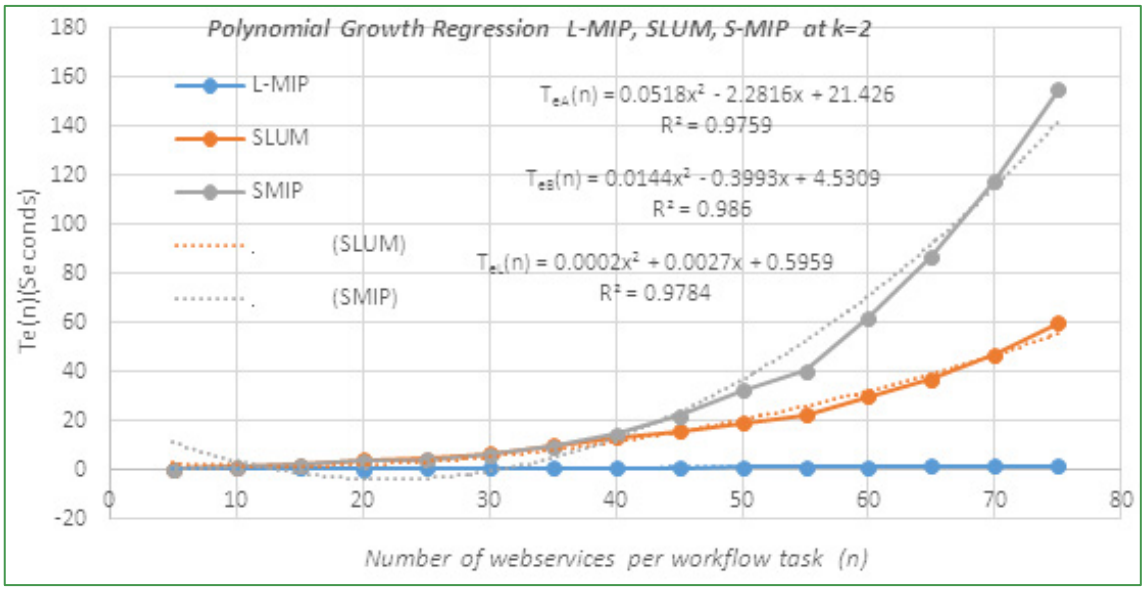

Figure 6. Polynomial Regression: Log $T_e(n)$ vs n. $T_e(n)$ stands for $T_{eA}(n)$, $T_{eL}(n)$, and $T_{eB}(n)$,

4.2.3. Measuring SLUM Expected Speedup (SES) w.r.t S-MIP

Based on the results, under the worst case analysis, both SLUM and S-MIP have an exponential growth. However, as can be seen from figure 3, the runtime growth of SLUM is evidently slower than that of S-MIP. Under exponential growth, we would like to determine the relative ratio of growth of SLUM w.r.t S-MIP. Applying equation (22) to the exponential equations in figure 4, we have $SES_e = 0.69e^{0.017n}$. Since $0.69e^{0.017n}$ is an increasing exponential function, intuitively, $0.69e^{0.017} \rightarrow \infty$, $n \rightarrow \infty$. Hence, this is an alternative proof that SLUM is much faster than S-MIP. By solving the inequality $0.69e^{0.017n} > 1$, we obtain $n_{CE} = 22$. $n_{CE} = 22$, suggests that SLUM starts overcoming S-MIP when the number of webservices per workflow task is more than 22. The result indeed is valid because, by examining the SIS values in table 2, we see that at *n=10* and *n=15*, SIS =1, and at *n=40*, SIS=1.09. By treating the SIS values at the intervening points at *n=20, n=30 and n=35* as outliers, one would expect the speedup value at *n=22* to be equal or more than 1. In addition, under polynomial growth functions from figure 6, applying L-Hospital's Rule gives SES =0.0518/0.0144 =3.8, implying that on average SLUM could be 3.6 times faster than S-MIP in the long run.

4.2.4. Measuring Empirical Relative Complexity of SLUM w.r.t S-MIP

We plot log t_{eB} vs log t_{eA} as shown in figure 7. The relationship log t_{eB} vs log t_{eA} seems to be strongly linear. We checked the significance of the relationship using the Microsoft Excel ToolPak plugin, and obtained a *p* value of 0.01 for the intercept and a *p* value of $3.85 * 10^{-13}$ for the X variable. These results indicate that the linear relationship is not only strong but statistically significant at a significance level of 0.05. The linear relationship allowed to us to compute the values of β_0 and β_1 in the equation (25). We obtain $\beta'_0 = 0.2506$ *and* $\beta_1 = 0.7833 \rightarrow t_{eB} = 1.28{t_{eA}}^{0.783} \rightarrow \beta_0 = 1.28$ and $\beta_1 = 0.7833$. Since $\beta_0 > 1$, we reject the null hypothesis in equation (27) and accept the alternative hypothesis of equation (28). Additionally, $\beta_1 < 1$ and therefore we reject the null hypothesis in equation (29) and accept the alternative hypothesis in equation (30). We thus conclude that S-MIP is 1.28 times faster than SLUM initially, but SLUM is more efficient than S-MIP asymptotically since $t_{eB} = {t_{eA}}^{0.783}$ for large enough t_{eA}.

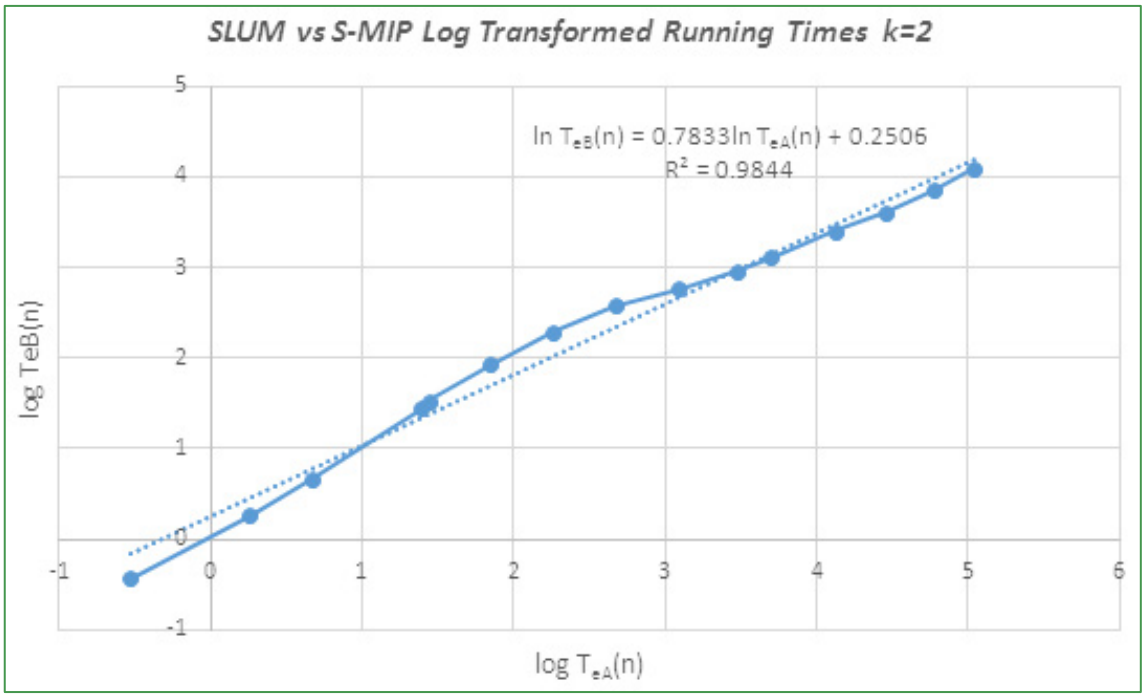

Figure 7. Log Transformed Empirical Running Times of SLUM and S-MIP

4.2. Discussions

The purpose of this research was to answer the research questions *RQ1*, *RQ2, RQ3 and RQ4* as stated in section 1.5. In this section we discuss to what extent the results obtained in section 4.2 answer each of the research questions.

4.3.1. Solution Quality

In response to *RQ1,* the results in section 4.1 indicate that the optimality ratio of SLUM is about 93% on average. We conclude that SLUM suboptimal relative to S-MIP. This finding confirms the early hypothesis of our recent work in [48]. The reason for this result is because, although SLUM considers all global constraints eventually, it does so in two layered steps, so that at one layer only global constraints related to the QoS attributes at that layer are considered. Thus, while SLUM guarantees global optimality within each layer, it does not guarantee the same at the *network* level because optimization at each layer does not take into account the constraints at the other layer.

To answer *RQ2,* we found that SLUM on average is significantly more optimal than L-MIP by approximately 5%. We draw the conclusion that SLUM on average yields more quality solutions than L-MIP. The reason why SLUM has a better solution quality performance could be qualitatively attributed to the fact that while SLUM considers global constraints albeit in two partial steps, L-MIP does not at all take into account global constraints, and thus, the chance of L-MIP ignoring potentially better webservices across the workflow are higher than SLUM. A possible quantitative explanation for the same observation is as follows: for each k by n workflow, L-MIP only considers the QoS matrices of n webservices at a time. Given that globally, the maximum number of possible solutions is n^k, for every optimization decision L-MIP takes, L-MIP ignores the QoS *of* $n^k{\_}n$ other possible solutions. SLUM on the other hand, at each layer, considers approximately only half of the QoS matrices per workflow task i.e recall that each worklow task is mathematically represented as matrix. Each matrix contains a set of n QoS vector, where each vector is a webservice. By considering only half of the QoS attributes at a time, SLUM only considers *n/2(*half of the QoS information) options per task. Thus at each layer, SLUM considers only $(n/2)^k$ possible options, while it ignores the remaining $(n)^k$- $(n/2)^k$ options. At a constant k, *i*t is possible to show that $(n)^k$- $(n/2)^k < n^k{\_}n$, $n>4$. For instance, let $n=6$ and $k=2$, $(n)^k$- $(n/2)^k = 27$, while $n^k{\_}n = 30$, and when $n=60$, $k=2$, $(n)^k$- $(n/2)^k = 2700$ and $n^k{\_}n$ =3540. Nevertheless, we also observed that in some instances, L-MIP, despite ignoring a larger number of candidate solutions, is able to produce solutions with better quality than SLUM. This isn't surprising because, in addition to the variety or set of candidate solutions considered, as illustrated in section 3.2.3.3, the statistical structure of the problem instances (QoS matrices) can affect the quality of the solution produced by one algorithm compared to the other. However, provided the problem instances are random in nature, for a very large number of problem instances, with monotonically increasing number of webservices per workflow task, there are more than 95% chances that SLUM will produce more quality solutions than L-MIP as indicated by the statistical tests.

Even if L-MIP performs worse than SLUM on average, we observed that its mean optimality ratio w.r.t S-MIP is

87%. This is still an impressive performance. The study conducted by [41] established that global planning MIP algorithms for webservice composition outperform local planning algorithms designed for the same task by 20% to 30% on average in terms of solution quality. This implies that, at the very best, local planning strategy has a mean optimality of 80% according to [41]. Although, our results report a figure slightly larger than 80%, our finding that L-MIP has 87% mean solution quality, is a reinforcement of the results by [41].

4.3.2. Running Time

Benatallah et al in [9] analytically demonstrated that the local planning strategy guarantees a solution in polynomial time. Using the results from section 4.2, we have quantitatively and numerically demonstrated that the running time of L-MIP has a polynomial upper bound, thus empirically reinforcing the analytic evidence. We have also shown that L-MIP is multiple times faster than both S-MIP and SLUM. This result is expected since the total number of candidate solutions evaluated by L-MIP is nk against a possible maximum of $2.(n/2)^k$ for SLUM and a possible maximum of $.(n)^k$ for S-MIP and thus the computational effort is smallest when using L-MIP compared to the other approaches.

We also determine that the running time of S-MIP is empirically bounded between polynomial and exponential growth – this implies that the running time of S-MIP is nondeterministic polynomial or superpolynomial. Although, Benatallah et al [9] had qualitatively noted that in some cases, the S-MIP global planning strategy could be exponential in running time, to the best of our knowledge, no quantitative proof exists that shows superpolynomial growth of MIP based global planning methods for the dynamic composite webservice selection problem. Thus the results complement the existing qualitative analysis such as those in [9]. Similarly, we establish that the running time of SLUM is superpolynomial. Indeed, this result provides further evidence that solving the composite webservice service selection problem remains *NP* hard to date as discussed in chapter 1 of this study.

Despite having found that both SLUM and S-MIP are superpolynomial theoretically, and therefore are equally good or equally bad in a theoretical sense, we did also report plausible practical performance differences between the two methods. Using the empirical relative complexity measure by Coffin et al [47], we found that for a small number of webservices per task, S-MIP is about 1.3 times faster than SLUM. For a large enough number of webservices per task, the SLUM is significantly faster than S-MIP since the empirical relative complexity coefficient of SLUM w.r.t to S-MIP =0.783<1. We also used L-Hospital's rule and limits theory to arrive at $n_{CE} = 22$ i.e beyond 22 webservices per worklow task, SLUM has a relative speedup larger than 1 and therefore faster than S-MIP. Using the same L-Hospital's rule we established that SLUM could be 3.6 times faster than S-MIP on average in the long run. These results could be attributed to the fact that initially, SLUM suffers the sequential overhead of having to formulate and instantiate the optimization problem twice on two sequentially partitioned problem instances, first one at the SCUM layer then later at the SPUM. The overhead is steadily overcome by the relative advantage of the layering as decomposition optimization approach as n grows larger and beyond 22, the sequential overhead is completely overcome and the superior performance of SLUM becomes apparent. This empirical result reinforces the theoretical claim that even when decomposition is formulated on sequential algorithms, relative performance speedups arise from the superlinear growth of the problems being solved [21].

Having $n_{CE} = 22$, means that virtual enterprise brokers having as few as 22 virtual enterprise service providers per workflow task, will find SLUM more efficient than S-MIP in meeting their dynamic webservice composition needs.

The finding that SLUM could be 3.6 times faster than S-MIP in the long run is not quite surprising. In section 1.3, we saw that the most global naïve planning algorithm would theoretically require n^k operations where k is the number of workflow tasks and n is the number of candidate services. By noting that each workflow task is mathematically a matrix of QoS vectors, applying a our two layered optimization model (SLUM) each of the matrix is split into two sub-matrices, one matrix passing through layer 1 optimization, while the other passing through layer two optimization ,though sequentially. Theoretically due to the two layer decomposition, we would expect a relative average speedup of $(1/2)^k = (1/2)^2 = 4$ (note that our evaluation involved two tasks only). However, the figure obtained is 3.6 is much smaller than the theoretical one because of the sequential computational overheads explained earlier.

It's quite surprising from the results in 4.2, to find that SLUM exhibits some moderate linear growth. Theoretically, this is unexpected and we could not account for the result. The result requires further investigation.

5. Conclusions

5.1. Summary of Research Contributions

The study compared the performance of the Service Layered Utility Maximization (SLUM) [48] method for the dynamic composite webservice selection problem within the context of virtual organizations, against the local planning strategy (L-MIP), and against the global mixed integer programming strategy in [9], S-MIP. The measures of performance were the CPU running time and the solution quality. Our first contribution is that through several statistical tests, we established that SLUM has an optimality ratio of 92.4% relative to S-MIP and that the solution

accuracy of SLUM is 5.4% better than L-MIP. The finding that L-MIP has 87% optimality ratio relative to S-MIP reinforces the early work by Ardagna et al [41]. We conclude that SLUM has a higher probability of satisfying with more accuracy, the webservice quality of service constraints demanded by service consumers. The results also support the existing theory that *layered network optimization* is relatively suboptimal compared to monolithic network optimization, as explained in theory of layering as optimization decomposition [24-27]. The same results, have demonstrated that S-MIP performs best in terms of solution quality. Therefore, in situations where there are no particular stringent requirements on timelines in service delivery, virtual enterprise brokers will find S-MIP most preferable among the three approaches.

Secondly, our empirical results show that L-MIP is several factors faster than both SLUM and S-MIP, for example at *n=50,* L-MIP is over 20 times faster than S-MIP and about 15 times faster than SLUM. In general, we show that L-MIP guarantees solutions in polynomial time. On the other hand, the results show that both SLUM and S-MIP exhibit superpolynomial growth in run time and therefore provide no guarantee of finding a solution in polynomial time. To the best of our knowledge, this is the first work to characterize the empirical complexity of S-MIP and SLUM as having a polynomial lower bound and exponential upper bound. This result has both theoretical and practical significance. To the research community, we provide further empirical evidence that the dynamic composite webservice selection considering global constraints, remains nondeterministic polynomial hard problem, alongside the analytic findings of [7], [10-12] and thus remains a significant problem deserving further research. We go beyond the abstract characterization based on empirical complexity, to show that even if the growth of the running time of both SLUM and S-MIP is superpolynomial, pragmatically, using statistical regression models, we demonstrate that S-MIP has an initial relative advantage over SLUM, such that below 22 webservices per workflow task, the method is 1.3 more efficient than SLUM. Otherwise, beyond *n=22,* we determine the empirical relative complexity of SLUM w.r.t to S-MIP as 0.783. Further, using L-Hospital's Rule from differential calculus, we show that SLUM is more than 3 times faster than S-MIP. The following additional theoretical contributions are made from these results. We provide an empirical proof that decomposition, even when applied to problems sequentially, eventually yields significantly more efficient solutions due to the super linearity of the complexity of computational problems as the problem size rises [21]. Concurrently, the results support the thesis of layering as decomposition [24-27], as a more efficient mathematical as well as architectural method for problems that inherently can be reformulated in multiple layers of abstraction- we provide the first proof where the composite webservice selection problem is concerned. Moreover, the results show that the relative expected efficiency gain of layering as decomposition with respect to non-layered is limited by the sequential overheads, hence achieving theoretical maximum expected speedup with respect to S-MIP may not be feasible.

Considering the foregoing, the overall and practical contribution is that for virtual enterprise brokers to gain maximum benefit from dynamic webservice composition, there is a need to combine the three techniques, given that none of the methods is adequate in all situations. This transforms to what Rice in [65] terms as *The Algorithm Selection Problem.* In this case, the question becomes, which of the three algorithms should the VEB use under what circumstances? Our contribution to this is that 1) In scenarios where there is no need for global constraints, L-MIP is the most ideal to technique to use especially in ultra-low latency webservice enabled collaborative applications such as online stock trading platforms. In such web applications, the tolerable waiting limit for end users is 2 seconds [13-14] ,[16-17] , 2) Below 22 webservices per task, the difference between SLUM and S-MIP is below 1 second. Where there are requirements for global constraints and strict requirements for 100% optimality, S-MIP is better than SLUM and L-MIP since SLUM does not guarantee global optimality whereas L-MIP lacks support for global constraints and at the same time is suboptimal 3), Where there is need to address global QoS constraints and the VEB has more than 22 service providers per workflow task SLUM is the best compromise with a little sacrifice on quality, 4) based on [48], if there are no strict requirements on timelines and on solution quality, and the target service consumers are average users, SLUM dominates over S-MIP and L-MIP because unlike the rest, SLUM does not require users to directly specify constraints on low technical parameters.

5.2. Ongoing Work

A surprising result that we could not account for is that SLUM exhibited modest linear growth. Although this behaviour is highly desirable, it has no theoretical basis so far. Our empirical analysis excluded the linear growth as a basis of benchmarking SLUM against S-MIP and L-MIP. As such, the conclusions still remain valid. However, we are performing more experiments using workflows having more than two tasks to investigate the result further.

REFERENCES

[1] Molina A. and Flores M., "A Virtual Enterprise in Mexico: From Concepts to Practice", Journal of Intelligent and Robotics Systems, 26: 289-302, 1999.

[2] Amit G., Heinz S .and David G. (2010). Formal Models of Virtual Enterprise Architecture: Motivations and Approaches,

PACIS 2010 Proceedings.

[3] Cammarimha M. and Arfsamanesh. (2007). A comprehensive modelling framework for collaborative networked organizations, Journal of Intelligent Manufacturing, Springer Publisher, Vol. 18 (5),pp-527-615.

[4] Arfsamanesh H. et al (2012). A framework for automated service composition in collaborative networks, FNWII: Informatics Institute, http://hdl.handle.net/11245/1.377/099.

[5] Dustdar S. and Wolfgang S. (2005). A Survey on Web Services Composition.

[6] Kuyoro Shade O. et al (2012). Quality of Service (QoS) Issues in Web services, International Journal of Computer Science and Network Security, Vol 12 (1), January, 2012.

[7] Mahboobeh M. and Joseph G.D (2011). Service Selection in Web service Composition. A comparative Review of Existing Approaches, Springer- Verlag Berlin, Heidelberg, 2011.

[8] Rajendran T. and Balasubramanie P. (2009). Analysis on the Study of QoS Aware Web services Discovery, Journal of Computing Vol. 1(2), December, 2009.

[9] Benatallah B. et al (2005). QoS-Aware Middleware for Web Services Composition. IEEE Transactions on Software Engineering, Vol. 30, No. 5, May 2005.

[10] Bin Xu et al (2011). Towards Efficiency of QoS driven semantic web service composition for large scale service oriented systems, Sringer, 211, DOI 10.1007/s11761-011-0085-8.

[11] Singh K.A (2012). Global Optimization and Integer Programming Networks. International Journal of Information and Communication Technology Research.

[12] Peter Bartalos, M´ariaBielikova (2011). Automatic Dynamic Web Service Composition: A Survey and Problem Formalization, Computing and Informatics Journal, Vol. 30, 2011, 793–827.

[13] Jacob Nielsen (1993). Usability Engineering, Morgan Kaufmann, 1st Edition, September, 1993.

[14] September 14, 2009 - Akamai Reveals 2 Seconds as the New Threshold of Acceptability for e Commerce Web Page Response Times http://www.akamai.com/html/about/press/releases/2009/press_091409.html, Last Accessed 4th April,2015.

[15] Alberto Savoia (2009). Webpage Response Time. Understanding and Measuring Performance Test Results. http://ericgoldsmith.com/wp-content/uploads/2009/02/web_page_response_time_101.pdf.

[16] http://www.nngroup.com/articles/response-times-3-important-limits/, updated 2014, Last accessed on 4th April, 2015.

[17] Nah 4. (2004). A Study on tolerable waiting time. How long are web users willing to wait? Behavior and Information Technology, Forthcoming.

[18] HC-L and K. Yoon (1981). Multiple Criteria Decision Making, Lecture Notes in Economics and Mathematical Systems, Springer Verlag. J Op. Res Soc. Vol 49(3), pp 237-252, March 1998.

[19] Toni Mancini, Piere Flener, Amir Hossein Monshi and Justin Pearson (2009). Constraint Optimization over Massive Databases in Proceedings of the 16th International Conference RCRA workshop (RCRA 2009).

[20] Seog Chan Oh, Dongwon Lee, and Soundar R. T. Kumara (2006). A comparative illustration of AI planning-based web services composition, ACM.

[21] Stephen Byod, Lin Xiao and Almir Mutapcic (2003). Notes on Decomposition Methods, Notes for EE3920, Stanford University, Autum available at 2003 http://web.stanford.edu/class/ee392o/decomposition.pdf, Last accessed 30th December, 2015.

[22] Matthew Kitching (2010). Decomposition and Symmetry in Constraint Optimization Problems, PhD Thesis, Graduate Department of Computer Science, University of Toronto.

[23] Kautz H. and Selman B. (1992). Planning as Satisfiability, available online at http://www.cs.cornell.edu/selman/papers/pdf/92.ecai.satplan.pdf, last accessed on 5th April, 2015.

[24] Chiang Mung. (2006). Layering as Optimization Decomposition, Electrical Engineering Department, Princeton University. Also available online athttp://www.ece.rice.edu/ctw2006/talks/ctw06-chiang.pdf.

[25] Chiang M. et al. (n.d). Layering as Optimization Decomposition. Current Status and Open Issues, Electrical Engineering Department, Princeton University.

[26] Chiang M. et al. Layering as Optimization Decomposition. Ten Questions and Answers, available at http://web.stanford.edu/class/ee360/previous/suppRead/read1/layer_1.pdf.

[27] Steve Low (2013). Scalable Distributed Control of Networks of DER, Computing & Math Sciences and Electrical Engineering, Caltech University.

[28] Kelly F.P, Maulloh A and Tan T (1998). Rate Control for Communication Networks. Shadow Prices, Proportional Fairness and Stability.

[29] Rao Jinghai and Xiaomeng Su (2004). A Survey of Automated Web Service Composition Methods.

[30] Shan S. and Gang G.G (2009). Survey of Modeling and Optimization Strategies for High Dimensional Design Problems with Computationally Expensive Black Box Functions, Springer Verlag, Published Online August, 2009.

[31] Kounev S., Gorton I and Sachs K. (Eds).SIPEW 2008, LNCS, 5119, 283-302, 2008, Springer Verlag.

[32] Shiang Chia Liu (2012). Applying Genetic Algorithm to Select Web services Based on Workflow Quality of Service, Journal of Electronic Commerce, Vol 13(2), 2012.

[33] Alrifai Mohammad (n.d) Distributed and Scalable QoS Optimization for Web services Composition, PhD Thesis, L3S Research Center, Leibniz University of Hannover, Germany.

[34] Amel Boustil, Nicholas Sabouret and Ramdane Maamri (2010). Web services Composition Handling User Constraints. A semantic approach.

[35] Mahdi Bakhshi and Seyyed Mohsen Hashemi (2012). User Centric Optimization for Constraint Web service Composition Using a Fuzzy Guided Genetic Algorithm System. International Journal on Webservice Computing Vol. 3, No 3. , September 2012.

[36] Mohammad Alifarai, Dimitrios Skoutas and Thomas Risse (2010). Selecting Skyline Services for QoS based Web service Composition, April 26-30, Raleigh NC, USA.

[37] Virginie G. et al (2013). A linear Program for QoS web service composition based on complex workflow. 2013.

[38] Shanliang Pan and Qinjiao Mao (2013). Case Study on Web services Composition Based on Multi-Agent System. Journal of Software, Vol. 8, No 4. April 2013.

[39] Fan Yan (2012). Global Optimization Method for Web services composition based on QoS, International Conference on Engineering and Business Management, 2012.

[40] Ngoko Y., Goldman A, and Milojicic D. (2013). Service Selection in Web service Compositions Optimizing Energy Consumption and Service Response.

[41] Ardagna, D. and B. Pernici, *Adaptive Service Composition in Flexible Processes.* IEEE Trans. on Software Eng., 2007.

[42] Rabelo R., Gusmeroli S. The ECOLEAD collaborative business infrastructure for networked organizations networked organizations. Pervasive collaborative networks PRO-VE 2008. Springer, New York, 2008.

[43] Web Applications Performance Symptoms and Bottlenecks Identification, www.agileload.com, last accessed 30th December 2015.

[44] Peter M. Broadwell (2004). Response time as a Performability Metric for Online Services, Report No. UCB//CSD-04-1324, Computer Science Division, University of California, Berkeley.

[45] Pervasive collaborative networks PRO-VE 2008. Springer, 33 (6): p. 369-384.

[46] Susan D. Urban and Le Gao. (xxxx). A Survey of Transactional Issues for Web Service, Composition and Recovery, Int. J. Web and Grid Services, Vol. x, No. x, xxx.

[47] Marie Coffin and Mathew J. Saltzman (2000). Statistical Analysis of Computational Tests of Algorithms and Heuristics, INFORMS Journal on Computing, Vol. 12, No. 1, Winter 2000.

[48] Abiud W. Mulongo, Elisha T. Omulo and William O. Odongo (2015). A Hierarchical Multilayer Service Composition Model for Global Virtual Organizations, Computer Science and Information Technology 3(4):91-104, 2015.

[49] Eugene Nudelman (2005). Empirical Approach to the Complexity of Hard Problems: PHD Thesis 2005, Stanford University.

[50] Holdger H. Hoos (2003). Introduction to Empirical Algorithmics

[51] Zemel Eitan (1981). Measuring the Quality of Approximate Solutions to Zero-One Programming Problems. Mathematics of Operations Research, Vol. 6, No. 3, August 1981, USA.

[52] Howel C. David (2013). Statistical Methods for Psychology, 8th Edition, ISBN-13: 978-1-111-83548-4

[53] Thomas Bartz-Beielstein and Mike Preu (n.d). Experimental Analysis of Optimization Algorithms: Tuning and Beyond, available at http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.301.2339&rep=rep1&type=pdf, last accessed on 22nd March, 2016

[54] Richard S. Barr et al (2001). Guidelines for Designing and Reporting on Computational Experiments with Heuristic Methods, March 16, 2001

[55] Goldsmith Fredrick Simon (2009). Measuring Empirical Computational Complexity, PhD Thesis, Department of Electrical Engineering and Computer Science, University of California at Berkeley, USA.

[56] Ravindra K. Ahuja and James B. Orlin (1996). Use of Representative Operation Counts in Computational Testing of Algorithms. Informs Journal on Computing, Vol. 8, Summer 1996.

[57] Holdger. H. Hoos (2009). A bootstrap approach to analysing the scaling of empirical run-time data with problem size. Technical Report TR 2009-16, Dept. of Computer Science, University of British Columbia, 2009.

[58] Holger. H. Hoos and T. (2014). On the empirical scaling of run-time for optimal solutions to the travelling salesman problem. European Journal of Operational Research, 238(1), 2014.

[59] Zongxu Mu and Holder. H. Hoos (2015). On the empirical time complexity of random 3-SAT at the phase transition, in the Proceedings of IJCAI, 2015.

[60] Holger H. Hoos and Zongxu Mu (2015). Empirical Scaling Analyser: An Automated System for Empirical Analysis of Performance Scaling, GECCO '15 July 11-15, 2015, Madrid, Spain, ACM ISBN 978-1-4503-3488-4/15/07, DOI: http://dx.doi.org/10.1145/2739482.2764898, last accessed on 22nd March 2016.

[61] Annay Levitin (2011). Introduction to the Design and Analysis of Algorithms, 3rd Edition.

[62] Xiaomei Zhu (2006). Discrete Two-Stage Stochastic Mixed-Integer Programs with Applications to Airline Fleet Assignment and Workforce Planning Problems, PhD Thesis, Department of Industrial Systems Engineering, Virginia Polytechnic Institute and State University.

[63] Ed Klotz, and Alexandra M. Newman (2012). Practical Guidelines for Solving Difficult Mixed Integer Linear Programs

[64] Rabelo J. Ricardo et al (2007). An Evolving Plug and Play Business Infrastructure for Networked Organizations. International Journal of on Information Technology and Management, 2007.

[65] Rice R. John (1976). The Algorithm Selection Problem. Computer Science Technical Reports, Report No. 75-76, Computer Science Department, Purdue, University