Adaptive Large Neighborhood Search Heuristics for Multi-tier Service Deployment Problems in Clouds

Anders N. Gullhav[a,*], Jean-François Cordeau[b], Lars Magnus Hvattum[a], Bjørn Nygreen[a]

[a]*Department of Industrial Economics and Technology Management, Norwegian University of Science and Technology, NO-7491 Trondheim, Norway*
[b]*HEC Montréal, 3000 chemin de la Côte-Saint-Catherine, Montréal, Canada, H3T 2A7*

Abstract

This paper proposes adaptive large neighborhood search (ALNS) heuristics for two service deployment problems in a cloud computing context. The problems under study consider the deployment problem of a provider of software-as-a-service applications, and include decisions related to the replication and placement of the provided services. A novel feature of the proposed algorithms is a local search layer on top of the destroy and repair operators. In addition, we use a mixed integer programming-based repair operator in conjunction with other faster heuristic operators. Because of the different time consumption of the repair operators, we need to account for the time usage in the scoring mechanism of the adaptive operator selection. The computational study investigates the benefits of implementing a local search operator on top of the standard ALNS framework. Moreover, we also compare the proposed algorithms with a branch and price (B&P) approach previously developed for the same problems. The results of our experiments show that the benefits of the local search operators increase with the problem size. We also observe that the ALNS with the local search operators outperforms the B&P on larger problems, but it is also comparable with the B&P on smaller problems with a short run time.

Keywords: metaheuristics, cloud computing, replication, local search

1. Introduction

An increasing proportion of enterprise and business software, such as customer management systems, email systems and time management applications, are run as web services in clouds through the software-as-a-service (SaaS) model. However, Marston et al. (2011) identify the lack of quality of service and availability guarantees as one of the major weaknesses of adopting cloud software services. Even though frameworks and software systems offering fault tolerance management in clouds have been proposed (Cully et al., 2008; Distler et al., 2011; Jhawar et al., 2013), there exist very few optimization models considering fault tolerance by introduction of redundancy (Avižienis et al., 2004) in the literature. Distler et al. (2011) present a fault tolerance approach based on active-passive replication, where passive backup replicas are run in a paused state, from which they can be activated rapidly. The passive replicas do not serve demand while being paused, and the replicas consume considerably less resources than corresponding demand-serving active replicas. We take these ideas into account when regarding the service deployment problem of a SaaS provider (SP). In this problem we consider decisions related to the replication of the SaaS services simultaneously with placement decisions. In previous work (Gullhav and Nygreen, 2015), we presented two mathematical models for the problem: one that considers service placement in a hybrid cloud, and another one that only considers placement in the private cloud of the service provider. We refer to Mell and Grance (2011) for definitions of the different types of clouds.

The SP offers a set of SaaS services, modeled as multi-tier services, to its clients. A multi-tier service is a service composed of multiple tiers that collaborate to deliver a service to the clients, and a typical example of a multi-tier

[*]Corresponding author

Email addresses: `anders.gullhav@iot.ntnu.no` (Anders N. Gullhav), `jean-francois.cordeau@hec.ca` (Jean-François Cordeau), `lars.m.hvattum@himolde.no` (Lars Magnus Hvattum), `bjorn.nygreen@iot.ntnu.no` (Bjørn Nygreen)

service is a three-tier web service composed of a web server, an application server and a database server. Figure 1 illustrates a three-tier service. When deployed in a cloud environment, each of the tiers, referred to as components of the service run in separate virtual machines (VMs). In turn, the VMs are placed on physical machines, which we refer to as nodes. In our work, we assume that the service provider owns and operates one or more data centers forming a private cloud, and can lease additional VMs in a public cloud when needed. Furthermore, the services of the SP are required to have a certain level of quality of service (QoS), as specified in service level agreements (SLAs), i.e., contracts between the SP and its clients. The QoS might be specified in terms of bounds on the performance, e.g., the response time, and bounds on the dependability, e.g., the availability or downtime. To obtain a satisfactory performance, each service component can be replicated into a number of load-balanced replicas (in separate VMs) that serve the clients in parallel. However, when one is to provide services run on a failure-prone infrastructure, such as the underlying hardware of clouds, one should take actions to limit the effects of the faults on the services. A key technique to make services fault-tolerant is standby redundancy. The concept of standby redundancy can be explained by the following slightly simplified but illustrative example. Consider a three-tier service with two load-balanced web server replicas, two load-balanced application logic replicas and a single database server. If the database server fails due to a fault, the service would obviously be counted as unavailable by the clients until the VM running the database server is restarted. Instead, if one of the two web server replicas fails due to a fault, there would only be one web server replica serving the clients until the second web server replica is restarted, and the clients would experience a lower performance (e.g., longer response times) in this time window. In both of these cases, running passive backup replicas (also denoted standby replicas) that could be activated faster than the time it takes to restart a replica would reduce the unavailability and improve the performance. In the following, the load-balanced replicas are denoted active replicas while the backup replicas are denoted passive replicas.

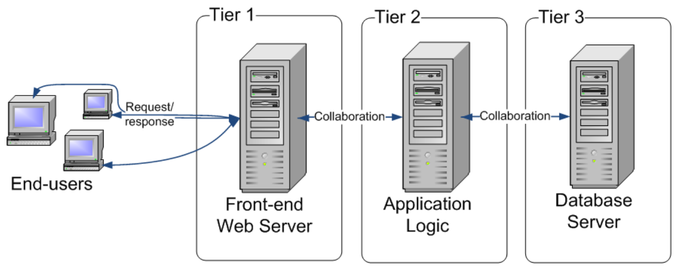

Figure 1: Illustration of a three-tier web service

The overall objective of the problem is to find the minimum cost deployment while respecting the QoS requirements of the SLA and other technical requirements, such as node resource capacities. In general, there is a non-linear relationship between the numbers of active and passive replicas of each component and the QoS of a service. To maintain a linear (mixed-integer) optimization model, Gullhav and Nygreen (2015) introduced a modeling structure called *replication patterns* that specifies a number of active and passive replicas of each component of a service such that the QoS requirements are satisfied. The replication level decisions are handled by selecting one replication pattern for each service, and hence, the details of the QoS requirements and models are not explicit in the optimization model, but instead handled when specifying replication patterns. By the use of the analytic queuing models of Gullhav et al. (2013) it is possible to evaluate the QoS of different replication patterns for a given service, and thus, give several replication patterns for each service as input to the model. The reason to specify several replication patterns for each service is that when a service provider offers multiple services and these services are deployed on the same infrastructure, the cheapest way to replicate the components of a service is dependent of how other services are replicated and deployed. This is because a cost-efficient packing of the nodes is dependent on how well the VMs of the components of the different services fit together, and the number of VMs of each component of each service is governed by the replication patterns. Therefore, the increased flexibility that arise by specifying several replication patterns can result in more cost-efficient deployment. Note that all replication patterns for a given service are *minimal*, meaning that if one removes one active or passive replica from any tier, the QoS will no longer be satisfactory.

The literature proposing related placement problems is discussed in our previous paper (Gullhav and Nygreen, 2015) and a recent survey on resource management in clouds is given by Jennings and Stadler (2015). Goudarzi and Pedram (2011) and Ardagna et al. (2012) propose resource allocation models for deployment of QoS-constrained multi-tier services. Their models do not concern backup replication and placement of backup replicas as very few models in the service placement literature do. However, Bin et al. (2011) propose a solution method for a placement problem of an infrastructure-as-a-service (IaaS) provider, where some VMs require one or more backup locations to which they can be migrated in case of a failure. Another problem related to ours is the redundancy allocation problem, where the goal is to find the minimum cost allocation of parallel components to different subsystems in series, while maintaining a reliability higher than a given level (Kuo and Wan, 2007).

In Gullhav and Nygreen (2015), we modeled the problem as a direct mixed-integer program (MIP). In addition, the problem was reformulated as a stronger pattern-based model. The latter formulation was solved by an a priori column generation algorithm, also called pre-generation, where a subset of the feasible patterns were given to the master problem in advance of the optimization. Furthermore, in Gullhav and Nygreen (2016), we proposed a branch and price (B&P) algorithm, where patterns were generated dynamically instead of a priori. The B&P algorithm outperformed the pre-generation algorithm.

The contribution of this paper is to introduce two novel adaptive large neighborhood search (ALNS) algorithms for two variants of the service deployment problem. In addition to destroy and repair operators, which are part of the standard ALNS framework, a novel feature of the algorithms is a local search layer on top of the repair operators. A key question we seek to answer in the computational study of this paper is what benefits the local search operators could bring to the ALNS. Another special feature of the proposed algorithms is a MIP-based repair operator, in addition to other heuristic insertion operators. Since the repair operators vary with respect to their time-performance trade-off, we score the operators according to both their performance and time consumption in the adaptive operator selection. Furthermore, the computational study also compares the speed and solution quality of the ALNS algorithms and the previously proposed B&P algorithm for the two versions of the service deployment problem.

The outline of the paper is as follows. In the next section, we present a description of the service deployment problem, and in Section 3, we repeat the direct MIP formulation of Gullhav and Nygreen (2015). In Section 4, we give a description of the components of the proposed ALNS algorithms. The computational study is presented and discussed in Section 5, before Section 6 concludes the paper. Appendix A gives a summary of the main mathematical symbols used in the mathematical formulations and the description of the ALNS, while some additional details from the computational experiments are shown in Appendix B.

2. Problem Description

Let $\mathcal{S}$ be the set of multi-tier services, and let $\mathcal{Q}_i$ denote the set of components of service $i \in \mathcal{S}$. For brevity, we will denote component $q \in \mathcal{Q}_i$ of service i as the pair (i, q). The VMs running the service components might run in a public cloud or on the set of nodes, $\mathcal{N}$, in the private cloud of the service provider, and each node has a set of limited resources, $\mathcal{G}$, e.g, CPU, memory, and storage. The nodes are assumed to be identical, and have resource capacities B_g for all resources $g \in \mathcal{G}$. When placed on a node, an active replica of the pair (i, q) consumes G^A_{iqg} resources of type g. The public cloud IaaS providers offer different VM types of a fixed capacity and cost to run the service components in the public cloud. As an example, Amazon Web Services (2015) offers several general purpose VM types, ranging from `micro` to `10xlarge`, with stepwise increases in capacity and cost. When placing an active replica of the pair (i, q) in the public cloud, this replica is run in the VM type that offers at least G^A_{iqg} resources for all g, and the cost of this VM type is denoted by C_{iq}. However, while active replicas can be run in the public cloud, we assume that support for passive replicas are only present on the nodes in the private cloud, and when run on the nodes in a passive state, the passive replicas require G^P_{iqg} ($< G^A_{iqg}$) resources. To ensure that passive replicas can be activated, each node that runs one or more passive replicas must maintain an unassigned pool of resources. One has to make a trade-off between cost and fault tolerance when setting the size of this pool since a small pool might make it impossible to activate a passive replica on the node, while a large pool will result in a large amount of unused resources in a failure-free situation. Here, the size of the pool of shared backup resources is set to be larger than the resources required to activate any of the passive replicas running on the node. In addition, the number of passive replicas run on a node is limited to E. Moreover, the replicas of the same service component are required to be run on different nodes. This policy is referred to as node-disjoint placement. Otherwise, a single node failure could bring down several replicas of the same component. In the following, B_g is assumed to be normalized to 1 for all resources and, hence, G^A_{iqg} and G^P_{iqg} are fractions of the node resource capacities.

The replication of the service components is done to obtain a certain level of performance and make the service fault tolerant. We do not consider a specific QoS measure, but instead use a method to check whether a given replication pattern, as introduced in Section 1, of a given service results in a tolerable QoS according to the SLA. Gullhav et al. (2013) propose a method that takes the number of active and passive replicas of each component of a service and the component's assigned resources as input, and outputs an approximate response time distribution. The method also assumes that the VMs fail according to a Poisson process, and takes this into account when computing the approximation. This method can be used here if the SLA specifies bounds on the mean or a percentile of the response time distribution of a service. We let $\mathcal{R}_i$ be the set of replication patterns of service i, and let R^A_{iqr} and R^P_{iqr} denote the number of active and passive replicas of the pair (i, q) in replication pattern $r \in \mathcal{R}_i$.

In the QoS guarantees, we do not account for the network latency. When the VMs of a service are placed in the same data center, or the private cloud, this latency can be neglected. When VMs placed in different data centers or clouds communicate, the latency should ideally be accounted for. However, doing this simplification makes our models much less complex. Nevertheless, we set an upper bound on the number of different services on a node to D, which drives different components of the same service to be run on the same nodes. In turn, this will reduce the amount of inter-node communication in the private cloud.

3. Direct MIP Formulation

The direct MIP formulation (Gullhav and Nygreen, 2015) uses binary variables w_{iqn} and v_{iqn} to indicate the placement of an active replica and a passive replica of component $q \in \mathcal{Q}_i$ of service $i \in \mathcal{S}$ on node $n \in \mathcal{N}$ in the private cloud, respectively. Furthermore, the integer variables t_{iq} are used to keep track of the number of active replicas placed in the public cloud. The binary variables y_{ir} indicate the selection of a replication pattern $r \in \mathcal{R}_i$ for service i, and we use the variables m_{ng} to represent the amount of resource of type g that is reserved for activation of passive replicas on node n. Lastly, the binary variables s_{in} indicate whether a replica belonging to service i is placed on node n, or not. With these definitions, the service deployment problem can be formulated as follows:

$$\min z = \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{iq} t_{iq} \tag{1}$$

subject to

$$\sum_{r \in \mathcal{R}_i} y_{ir} = 1 \quad \forall i \in \mathcal{S} \tag{2}$$

$$\sum_{n \in \mathcal{N}} w_{iqn} + t_{iq} - \sum_{r \in \mathcal{R}_i} R^A_{iqr} y_{ir} = 0 \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i \tag{3}$$

$$\sum_{n \in \mathcal{N}} v_{iqn} - \sum_{r \in \mathcal{R}_i} R^P_{iqr} y_{ir} = 0 \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i \tag{4}$$

$$w_{iqn} + v_{iqn} - s_{in} \leq 0 \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i, \forall n \in \mathcal{N} \tag{5}$$

$$\sum_{i \in \mathcal{S}} s_{in} \leq D \quad \forall n \in \mathcal{N} \tag{6}$$

$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} v_{iqn} \leq E \quad \forall n \in \mathcal{N} \tag{7}$$

$$m_{ng} - (G^A_{iqg} - G^P_{iqg})v_{iqn} \geq 0 \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i, \forall n \in \mathcal{N}, \forall g \in \mathcal{G} \tag{8}$$

$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} G^A_{iqg} w_{iqn} + \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} G^P_{iqg} v_{iqn} + m_{ng} \leq 1 \quad \forall n \in \mathcal{N}, \forall g \in \mathcal{G} \tag{9}$$

$$m_{ng} \geq 0 \quad \forall n \in \mathcal{N}, \forall g \in \mathcal{G} \tag{10}$$

$$w_{iqn} \in \{0,1\} \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i, \forall n \in \mathcal{N} \tag{11}$$

$$v_{iqn} \in \{0,1\} \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i, \forall n \in \mathcal{N} \tag{12}$$

$$s_{in} \in \{0,1\} \quad \forall i \in \mathcal{S}, \forall n \in \mathcal{N} \tag{13}$$

$$y_{ir} \in \{0,1\} \quad \forall i \in \mathcal{S}, \forall r \in \mathcal{R}_i \tag{14}$$

$$t_{iq} \in \mathbb{Z}_+ \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i \tag{15}$$

The objective function (1) minimizes the total cost of placing replicas in the public cloud, and the equalities (2) ensure that one replication pattern is selected for each service. The two sets of equalities (3) and (4) establish the relation between the placement variables, w_{iqn}, t_{iq} and v_{iqn}, and the replication pattern variables y_{ir}. The constraints (3) ensure that given a selection of a replication pattern r for a service i (i.e., $y_{ir} = 1$), R^A_{iqr} active replicas of component q are placed either in the private or in the public cloud. The constraints (4) ensure the same for passive replicas, but confine the placement to the private cloud. The rest of the constraints model the technical requirements related to the placement in the private cloud. Specifically, the inequalities (5) take care of the requirement specifying that the replicas of the same pair (i, q) should be placed on different nodes, and at the same time force s_{in} to take value 1, as long as there is at least one replica from service i deployed on n. Moreover, constraints (6) and (7) put upper bounds on the number of different services, and the number of passive replicas on each node, respectively. The resource capacities of the nodes are handled by constraints (9), where the first and second terms account for the resources assigned to the active and passive replicas deployed on the node, and the third term accounts for the resources reserved for activation of passive replicas, which is set by the inequalities (8).

We now want to consider the special case where the number of nodes in the private cloud is large enough to run all replicas privately. As opposed to the *hybrid cloud model* (1)-(15) above, we refer to this model as the *private cloud model*. The objective function for this case might consist of several cost components, but we have chosen to focus on the power consumption of the nodes in the private cloud. In data centers, the power consumption of a node in an idle state is significant and can be as large as 70 % of the peak power consumption (Beloglazov et al., 2011). Hence, a common strategy, also applied in VM placement models in the literature, is to minimize the number of nodes used for placement. Herein, we also implement this objective, and introduce the binary variables u_n indicating whether node n is turned on and used for placement, or not. The hybrid cloud model (1)-(15) is then modified by replacing the objective function by (16). In addition, the active deployment equalities (3) are reduced to the equalities (17). We also need to prevent nodes that are turned off from being used, which is achieved by replacing constraints (9) by constraints (18). Finally, the variable definitions (19) are also added to the private cloud model, such that the private cloud model is formulated as the problem of minimizing (16) subject to (2), (17), (4)-(8), (18), (10)-(14), and (19).

$$\min z = \sum_{n \in \mathcal{N}} u_n \tag{16}$$

$$\sum_{n \in \mathcal{N}} w_{iqn} - \sum_{r \in \mathcal{R}_i} R^A_{iqr} y_{ir} = 0 \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i \tag{17}$$

$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} G^A_{iqg} w_{iqn} + \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} G^P_{iqg} v_{iqn} + m_{ng} - u_n \leq 0 \quad \forall n \in \mathcal{N}, \forall g \in \mathcal{G} \tag{18}$$

$$u_n \in \{0,1\} \quad \forall n \in \mathcal{N} \tag{19}$$

4. The Adaptive Large Neighborhood Search

The Adaptive Large Neighborhood Search (ALNS) is an extension of the Large Neighborhood Search (LNS) metaheuristic framework presented by Shaw (1997), and is also related to the ruin and recreate principle of Schrimpf et al. (2000). The ALNS was first proposed by Ropke and Pisinger (2006), and has been used in many applications since then. Pisinger and Ropke (2010) give an overview of the LNS and ALNS algorithms, and review some of the literature on applications of the algorithms.

The basic principle of the ALNS and LNS is to iteratively destroy and repair a solution, and accept the new solution as the incumbent if some criterion is met. A simple acceptance criterion is to only accept improving solutions, while a criterion commonly used in the literature is the simulated annealing (SA) acceptance criterion. The ALNS extends and deviates from the LNS by including several destroy operators and repair operators in the search for improving solutions. That is, in each iteration one destroy operator and one repair operator are chosen as the neighborhood operators. The classical destroy operators include random destroy, worst destroy and related destroy (Pisinger and Ropke, 2010). Generally, the first type of operator consists of partly destroying a solution or removing solution components at random, while the second operator type consists of removing the parts of the solution that are considered bad or critical. The last type of operator tries to remove solution components that are related in some way. The two last operators require, respectively, a way to measure which solution aspects that are bad and related. These measures and the algorithmic identification of the types of solution components to destroy are problem specific. All destroy operators used herein can be seen as problem specific implementations of one the classical operators types. Moreover, typical repair operators include greedy or regret-based constructive heuristics such as the ones by Ropke and Pisinger (2006), as well as exact algorithms as used by Shaw (1997).

The algorithms proposed in this paper also adopt the ideas of the Iterated Local Search (ILS) paradigm (Lourenço et al., 2010). The search process of ILS is based on iteratively improving a solution by local search until a local optimum is found, and then perturbing the solution before continuing with local search from the modified solution. The perturbation works as the diversification mechanism, and should be designed so that the search process manages to escape local optima. The local search works as the intensification mechanism. To increase the intensification of the ALNS, we have implemented different local search operators that are called in every iteration after the solution is repaired.

In addition to the inclusion of local search operators, one of the repair operators is based on solving a MIP, and thus, the ALNS can be seen as a hybridization of different optimization algorithms. There are many examples in the literature where ALNS or LNS, or more generally, metaheuristics, have been successfully hybridized with exact solution methods, such as general-purpose MIP solvers. Some hybrid approaches are reviewed by Raidl and Puchinger (2008). Conversely, there are few works applying simple local search operators in conjunction with LNS or ALNS. Parragh and Schmid (2011) propose an LNS for a dial-a-ride problem that uses dual information provided by solving a reduced version of the problem with column generation, and in addition at every 1000 LNS iterations they generate new columns by performing swaps between existing columns. Since the local search is used in only a few iterations, the local search has a less central role than in the algorithms proposed herein, where it is used in every iteration. Moreover, in Parragh and Schmid (2013), the same authors propose another type of hybridization for the same problem. Their hybridization is based on column generation, where the columns are generated both by LNS and Variables Neighborhood Search (VNS) (Hansen and Mladenović, 2001). The LNS is used in some iterations to search for new columns by destroying and repairing columns of a solution, while the local search-based VNS is used to search for new columns based on inserting and removing elements, or swapping elements between columns.

Algorithm 1 shows a pseudocode that gives an overview of the proposed ALNS metaheuristics. The sets of destroy, repair and local search operators are denoted O^D, O^R, and O^L, respectively. The initial feasible solution σ is created based on an initial assignment of replication patterns and a placement of the replicas in the private and public clouds. The construction of the initial solution is explained in detail in Section 4.6. While one attempts to change the replica placement in every iteration of the ALNS, the replication pattern assignment is only changed at some iterations. In Line 3, the destroy operator θ, repair operator ρ, and local search operator λ to be used in the current iteration are biasedly selected using the vectors of weights $\mathbf{\Psi}^D$, $\mathbf{\Psi}^R$, and $\mathbf{\Psi}^L$. In Line 5, the destroy, repair and local search operators are called sequentially. In Line 12, the weights of the operators are updated based on quality of the new solution, σ. Lines 3 and 12 are discussed in more detail in Section 4.5, while the different destroy, repair and local search operators are described in Sections 4.2, 4.3, and 4.4, respectively.

Algorithm 1 Pseudocode of the proposed ALNS metaheuristics

```
Require: a feasible solution $\sigma$ //$\bar{\sigma}$ is the incumbent solution and $\hat{\sigma}$ is the best found solution
 1: $\eta = 0; \bar{\sigma} = \sigma; \hat{\sigma} = \sigma; \mathbf{\Psi}^D = [1,\dots,1]^\intercal; \mathbf{\Psi}^R = [1,\dots,1]^\intercal; \mathbf{\Psi}^L = [1,\dots,1]^\intercal$
 2: repeat
 3:     select destroy, repair and local search operators $\theta \in O^D$, $\rho \in O^R$ and $\lambda \in O^L$ using scores $\mathbf{\Psi}^D$, $\mathbf{\Psi}^R$ and $\mathbf{\Psi}^L$
 4:     $\eta = \eta + 1$ //increment iteration counter
 5:     $\sigma = \lambda(\rho(\theta(\bar{\sigma})))$ //call the destroy, repair and local search operators sequentially
 6:     if accept($\sigma$,$\bar{\sigma}$) then
 7:         $\bar{\sigma} = \sigma$
 8:         if $\sigma$ is better than $\hat{\sigma}$ then
 9:             $\hat{\sigma} = \sigma$
10:         end if
11:     end if
12:     update($\mathbf{\Psi}^D$,$\mathbf{\Psi}^R$,$\mathbf{\Psi}^L$)
13: until $\eta = I$ //stop when reaching the maximum number of iterations, I
14: return $\hat{\sigma}$
```

4.1. Cost Functions and Acceptance Criteria

The acceptance criterion used in Line 6 of Algorithm 1 is based on the classical SA criterion: with a temperature τ and cost function $c(\sigma)$, a new solution σ is accepted with probability $\min\{1, e^{\frac{-(c(\bar{\sigma})-c(\sigma))}{\tau}}\}$, where $\bar{\sigma}$ is the current solution. Starting from an initial temperature τ_0, the temperature is gradually reduced every iteration by multiplying τ by a factor $\alpha \in (0,1)$. Thus, the probability of accepting a new solution that is worse than the incumbent is gradually reduced.

In the ALNS for the hybrid cloud model, the objective function (1) is a natural choice as a cost function. However, the objective function of the private cloud model (16) leads to a fitness landscape consisting of large plateaus, and it would not be effective to use this function to drive the heuristic search. This problem is also observed in vehicle routing problems that minimize the number of vehicles as the primary objective. Pisinger and Ropke (2007) allow their ALNS for vehicle routing problems to work with infeasible solutions when minimizing the number of vehicles. Specifically, they allow some requests not to be served and penalize the unserved requests in the cost function. If a new solution with no unserved requests is found, the number of vehicles used in the future solutions is reduced by one. Similarly, we allow the ALNS to examine infeasible solutions by relaxing the node resource constraints (18), and penalize the surplus resource usage as the only term in the cost function. Thus, the set of nodes that can be used for placement is gradually reduced, and we denote the set of nodes available for placement for a solution σ as $\mathcal{N}(\sigma)$. Formally, with e_{ng} defined as the surplus usage of resource type $g \in \mathcal{G}$ on node $n \in \mathcal{N}(\sigma)$, the cost function of the ALNS in the private cloud case is given by:

$$c(\sigma) = \sum_{n \in \mathcal{N}(\sigma)} \sum_{g \in \mathcal{G}} e_{ng} \tag{20}$$

If a solution with $c(\sigma) = 0$ is found, the number of nodes allowed in the next solution is reduced by one. We denote solutions of the private cloud model that violate only the node resource constraints (18) as *over-utilized*, and include them in the set of feasible solutions.

4.2. Destroy Operators

We describe seven methods to destroy a solution. The first five consider only destruction of parts of the private cloud, the sixth method considers destruction of parts of both the private and public cloud, while the last method only removes replicas from the public cloud. The destruction of the private cloud is done in several ways, either by removal of all replicas on a subset of the nodes, removal of all active replicas on a subset of the nodes, removal of all passive replicas on a subset of the nodes, or removal of all replicas of a subset of the services. All methods take the current solution σ and a parameter $\delta \in (0, 1]$, the degree of destruction, as input. In each iteration, the number of replicas to remove is chosen randomly based on the value of δ. In addition, all but the first method (Random Node Removal) have a parameter $\gamma \in \mathbb{R}_+$ that controls the degree of randomization in the destruction. For brevity, we define $Q_A(n)$, respectively $Q_P(n)$, as the set of pairs (i, q) which have an active replica, respectively a passive replica, placed on node $n \in \mathcal{N}(\sigma)$ in the current solution σ.

To ease the exposition, we use the terms *displace* and *displacement* with the meaning of moving a replica either from a node in the private cloud or the public cloud to a set of unplaced active replicas or unplaced passive replicas.

4.2.1. Random Node Removal

This method randomly selects h nodes for removal, that is, all active and passive replicas at these h nodes are displaced. The number of nodes h to destroy is randomly drawn according to a uniform distribution among the integers in the interval $[2, \lfloor\delta|\mathcal{N}(\sigma)|\rfloor]$.

4.2.2. Active Replica Removal

The idea of this operator is to select h nodes and displace all active replicas on these nodes, while preserving the passive replicas. Instead of selecting the h nodes randomly, we want to bias the selection towards nodes having a specific feature, denoted by ϕ. The feature can either be low resource utilization as defined in equation (21) below, or low average public cloud cost of the active replicas running on the node as defined in equation (22). Only the first of these features is used in the ALNS for the private cloud model. Feature ϕ_A only considers the active part of the resource utilization of the nodes, that is, keeping all else fixed, it measures how well the node is utilized by active replicas.

$$\phi_A(n) = |\mathcal{G}|^{-1} \sum_{g \in \mathcal{G}} \frac{\sum_{(i,q)\in Q_A(n)} G^A_{iqg}}{1 - \sum_{(i,q)\in Q_P(n)} G^P_{iqg} - m_{ng}} \tag{21}$$

$$\phi_C(n) = |Q_A(n)|^{-1} \sum_{(i,q)\in Q_A(n)} C_{iq} \tag{22}$$

The pseudocode of the Active Replica Removal, where the node selection is biased towards nodes with feature ϕ, is shown in Algorithm 2. The randomness of the algorithm is controlled by γ, and setting $\gamma = 1$ makes the node selection in Line 4 completely random. The parameter h is randomly and uniformly drawn among the integers in $[2, \lfloor 2\delta|\mathcal{N}(\sigma)|\rfloor]$.

Algorithm 2 Pseudocode of the Active Replica Removal operator

```
Require: a solution $\sigma$, $h \in \mathbb{N}$, $\gamma \in \mathbb{R}_+$
1: let $\boldsymbol{N}$ be a vector of the nodes in solution $\sigma$, sorted in ascending order of feature $\phi$
2: while $h > 0$ do
3:     $\pi$ = random number $\in [0, 1)$
4:     select an element $n$ at position $\lfloor \pi^\gamma |\boldsymbol{N}| \rfloor$ of $\boldsymbol{N}$
5:     displace all active replicas of node $n$, and remove $n$ from $\boldsymbol{N}$
6:     $h = h - 1$
7: end while
8: return $\sigma$
```

4.2.3. Passive Replica Removal

The Passive Replica Removal operator only differs from the Active Replica Removal in Algorithm 2 at Line 5 where it displaces all passive replicas of node n, instead of all active replicas. In addition, the node vector is sorted in ascending order of the feature represented by ϕ_P, as defined in equation (23) below. Nodes running passive replicas with diverging requirements for the amount of the shared backup resource m_{ng}, and nodes running few passive replicas get a low value on ϕ_P. The rationale of this destroy operator is that it is beneficial that as many passive replicas as possible (bounded by E) share the reserved backup resources, and that the passive replicas running on the same node have quite similar resource requirements when being activated. The number of nodes, h, is drawn in the same way as in the active replica removal method.

$$\phi_P(n) = |\mathcal{G}|^{-1} \sum_{g \in \mathcal{G}} \frac{\sum_{(i,q) \in Q_P(n)} (G^A_{iqg} - G^P_{iqg})}{m_{ng} E} \tag{23}$$

4.2.4. Worst Node Removal

Like Random Node Removal, the Worst Node Removal operator selects h nodes and displaces all replicas running on the nodes. Thus, Algorithm 2 is adapted by changing Line 5 such that all replicas on the selected node are displaced. In this operator, the vector of nodes is ordered according to the feature ϕ_W as defined in (24) below. As long as there are passive replicas on the node, ϕ_W is a weighted combination of ϕ_A and ϕ_P, using weight parameter $\beta_W \in (0, 1)$. In the private cloud model, for a node n with over-utilized resources, i.e., $e_{ng} > 0$ for at least one g, ϕ_A takes a value larger than 1. In order not to favor this, we put an upper bound of 1 on ϕ_A.

$$\phi_W(n) = \begin{cases} (1 - \beta_W) \min\{1, \phi_A(n)\} + \beta_W \phi_P(n) & \text{if } Q_P(n) \neq \emptyset \\ \phi_A(n) & \text{otherwise} \end{cases} \tag{24}$$

4.2.5. Over-utilized Node Removal

This operator is quite similar to Worst Node Removal, but is specifically designed for use in the private cloud model. It biases the node selection primarily towards nodes with large over-utilization, and secondarily towards nodes with large under-utilization. This is achieved by sorting the node vector according to feature ϕ_O, given in (25), where we define $G_g(n)$ according to equation (26), that is, as the current usage of resource g at node n. Observe that the expression in (25) is negated, and that the first term, the over-utilization, is given a large weight Ω_O, such that a node, say n_1, with the smallest possible over-utilization is ordered before a node, say n_2, with the largest possible under-utilization, i.e., $\phi_O(n_1) < \phi_O(n_2)$.

$$\phi_O(n) = -\sum_{g \in \mathcal{G}} \Big(\Omega_O \max\{0, G_g(n) - 1\} + \max\{0, 1 - G_g(n)\} \Big) \tag{25}$$

$$G_g(n) = \sum_{(i,q) \in Q_A(n)} G^A_{iqg} + \sum_{(i,q) \in Q_P(n)} G^P_{iqg} + m_{ng} \tag{26}$$

4.2.6. Related Service Removal

All the destroy operators considered so far select nodes, randomly or according to some feature, and displace all or some of the replicas on the selected nodes. The idea of the related service removal is to look for similarities among the worst nodes, specifically by considering the services that are present on the worst nodes. The operator is illustrated in Algorithm 3. Similar to the worst node removal, the vector of nodes is sorted according to feature ϕ_W, and h_N nodes are selected and added to a set of the worst nodes. However, instead of displacing the replicas of these nodes, the operator finds the h_S services that are represented with the most replicas at the worst nodes. Then all replicas of the h_S services are displaced. The parameters h_N and h_S are randomly and uniformly drawn among the integers in the intervals $[2, \lfloor \delta |\mathcal{N}(\sigma)| \rfloor]$ and $[1, \lfloor \delta |\mathcal{S}| \rfloor]$, respectively.

Since this operator displaces complete services, it is simple for the subsequent repair operator to change the replication pattern of the services involved. While this might be possible for repair operators following other destroy operators as well, we have not considered this. Therefore, the replication patterns can only change after calling the Related Service Removal.

Algorithm 3 Pseudocode of the Related Service Removal operator

```
Require: a solution $\sigma$, $h_N \in \mathbb{N}$, $h_S \in \mathbb{N}$, $\gamma \in \mathbb{R}_+$
 1: let $\boldsymbol{N}$ be a vector of the nodes in solution $\sigma$, sorted in ascending order of feature $\phi_W$
 2: let $\mathcal{N}_W = \emptyset$ be the set of the worst nodes
 3: while $h_N > 0$ do
 4:     $\pi$ = random number $\in [0, 1)$
 5:     select an element $n$ at position $\lfloor \pi^\gamma |\boldsymbol{N}| \rfloor$ of $\boldsymbol{N}$
 6:     remove $n$ from $\boldsymbol{N}$, and add $n$ to $\mathcal{N}_W$
 7:     $h_N = h_N - 1$
 8: end while
 9: let $\mathcal{S}_W$ be the set of services with cardinality $h_S$ that are represented with most active and passive replicas on
    the nodes in $\mathcal{N}_W$
10: displace all replicas of the services in $\mathcal{S}_W$, from both the private cloud and the public cloud
11: return $\sigma$
```

4.2.7. Public Cloud Destruction

This operator, outlined in Algorithm 4, is the sole operator that only considers destruction of the public cloud placement. The parameter h, the number of active replicas to displace, is uniformly drawn among the integers in the interval $[5, \lfloor \delta_C |\boldsymbol{Q}_C(\sigma)| \rfloor]$, where δ_C is the degree of destruction, and $\boldsymbol{Q}_C(\sigma)$ is a vector of the active replicas currently placed in the public cloud. The displacement of replicas is biased towards replicas with high cost, C_{iq}, and the bias is still controlled by the parameter γ.

Algorithm 4 Pseudocode of the Public Cloud Destruction operator

```
Require: a solution $\sigma$, $h \in \mathbb{N}$, $\gamma \in \mathbb{R}_+$
 1: let $\boldsymbol{Q}_C$ be a vector of the active replicas $(i, q)$ currently placed in the public cloud in solution $\sigma$, sorted in
    descending order of public cloud cost $C_{iq}$
 2: while $h > 0$ do
 3:     $\pi$ = random number $\in [0, 1)$
 4:     select an element $(i, q)$ at position $\lfloor \pi^\gamma |\boldsymbol{Q}_C| \rfloor$ of $\boldsymbol{Q}_C$
 5:     displace active replica $(i, q)$ from the public cloud
 6:     $h = h - 1$
 7: end while
 8: return $\sigma$
```

This operator is only used in the ALNS for the hybrid cloud model, and it is used in combination with one of the other operators that consider destruction of the private cloud, except from the Related Service Removal. When this operator is called in combination with the Random Node Removal, $\gamma = 1$, that is, the displacement of replicas from the public cloud is completely random; otherwise γ takes the same value as the other destroy operators.

4.3. Repair Operators

We use three types of repair operators. The first two are fast and insert unplaced replicas one at a time, based on simple measures. They do not concentrate on rebuilding the nodes individually, like a strategy that solves several knapsack problems in sequence, but consider every node as a potential point of placement, as long as the insertion of a replica at the node is feasible. The last repair operator is based on solving a reduced version of the direct MIP models, where a large number of the variables is fixed to integer values. Even though the MIP is greatly reduced, the operator is more time-consuming than the insertion operators. However, if one gives the operator enough time, it will find the optimal way to repair the destroyed solution.

4.3.1. *Greedy Insertion*

The Greedy Insertion operator is a fast and simple constructive heuristic, which iteratively places the unplaced replicas on the nodes guided by a cost measure. Specifically, for the ALNS of the hybrid cloud model, if no more active replicas can be placed on the nodes, the rest of the unplaced active replicas are placed in the public cloud. However, if there are remaining unplaced passive replicas, the operator returns without any feasible solution. Even though the ALNS of the private cloud model allows over-utilized nodes in the solutions, it is possible that the greedy insertion cannot find a feasible placement for all unplaced active or passive replicas, and thus, returns without any feasible solution.

The cost measures used to guide the greedy insertion heuristic in the ALNS for the hybrid cloud model and the private cloud model differ. This is natural since the objective functions in the mathematical formulations also differ. Furthermore, the cost measures used to guide the insertion of active replicas and passive replicas differ, and they are not directly comparable. Therefore, the Greedy Insertion heuristic first tries to insert all passive replicas, and then focuses on inserting all active replicas.

In the hybrid cloud model, we let $\xi_P(i,q,n)$, defined in (27), denote the cost of inserting a passive replica of service component pair (i,q) at node n. The cost accounts for the absolute deviation between the current reserved backup resources, m_{ng}, at the node and the replica's requirement for backup resources. However, if the insertion is infeasible, we set $\xi_P(i,q,n) = \infty$. Analogously, we define $\xi_A(i,q,n)$, according to (28), as the cost of inserting an active replica of pair (i,q) at node n. The first term of (28), which is given a large weight Ω_C, accounts for the public cloud cost that would incur if the replica is not placed in the public cloud, while the second term accounts for the residual resource slack at the node after insertion (less is better). Since active replicas with large public cloud cost should have a smaller insertion cost $\xi_A(i,q,n)$, we subtract C_{iq} from a constant $\bar{C}$ which is greater than $\max_{(i,q)} C_{iq}$. Recall that $G_g(n)$ denotes the current resource usage at node n, as defined in (26).

$$\xi_P(i,q,n) = \sum_{g\in\mathcal{G}} |m_{ng} - (G^A_{iqg} - G^P_{iqg})| \tag{27}$$

$$\xi_A(i,q,n) = \Omega_C(\bar{C} - C_{iq}) + \sum_{g\in\mathcal{G}} \left(1 - G_g(n) - G^A_{iqg}\right) \tag{28}$$

The cost measures are adapted to (29) and (30) in the ALNS for the private cloud model. Since the overall cost function used by the acceptance criteria (see Section 4.1) concerns the amount of over-utilized resources, both cost measures below take this into account. We let $\Delta e^P_g(i,q,n)$, respectively $\Delta e^A_g(i,q,n)$, denote the increase in over-utilization (of resource type g) when a passive, respectively an active, replica of pair (i,q) is inserted on node n; this increase in over-utilization is given a large weight Ω_I in the cost measures.

$$\xi_P(i,q,n) = \Omega_I \sum_{g\in\mathcal{G}} \Delta e^P_g(i,q,n) + \sum_{g\in\mathcal{G}} |m_{ng} - (G^A_{iqg} - G^P_{iqg})| \tag{29}$$

$$\xi_A(i,q,n) = \Omega_I \sum_{g\in\mathcal{G}} \Delta e^A_g(i,q,n) + \sum_{g\in\mathcal{G}} \left(1 - G_g(n) - G^A_{iqg}\right) \tag{30}$$

In a greedy fashion, the operator first selects the passive replica among all unplaced replicas with minimum cost, and inserts this replica on its minimum cost node. This insertion process repeats until all passive replicas are inserted, or there are no more feasible insertions to do. In the latter case, the operator would return without a feasible solution. Assuming that all passive replicas were placed, the operator considers the unplaced active replicas in the same way. In the ALNS for the hybrid cloud model, if there are unplaced active replicas left after the insertion process, these replicas are placed in the public cloud. On the other hand, in the private cloud model the operator would return without a feasible solution unless all active replicas are inserted.

4.3.2. *Regret Insertion*

The Regret Insertion is designed to be less myopic than the Greedy Insertion in the insertion strategy. The operator extends the Greedy Insertion by being guided by a regret value, or look-ahead value, instead of the cost measure directly. For a passive (active) replica of pair (i, q), let n_{iqk} be the node with kth lowest cost according to cost measure $\xi_P(i, q, n)$ ($\xi_A(i, q, n)$). Moreover, the regret-k value $\zeta_{iq}(k)$ for a passive replica is defined as in (31). The regret-k value for active replicas is defined analogously using the cost measures $\xi_A(i, q, n)$ instead.

$$\zeta_{iq}(k) = \sum_{j=2}^{k} \Big(\xi_P(i, q, n_{iqj}) - \xi_P(i, q, n_{iq1})\Big) \tag{31}$$

The insertion procedure of the Regret Insertion is similar to that of the Greedy Insertion, except that instead of selecting the replica with lowest cost of insertion in each iteration, the replica with the *maximum* regret value is selected. If a replica has fewer than k nodes where an insertion is feasible, the regret value of the replica will be ∞. Hence, the replica is selected for insertion early in the process. This means that regret insertion operators with high k have greater probability of finding a feasible solution than both regret insertion operators with low k and the Greedy Insertion.

4.3.3. MIP Insertion

The MIP Insertion is based on the idea of calling a general-purpose solver on the MIP models presented in Section 2, where a part of the variables is fixed. In a given iteration of the ALNS, the fixed variables correspond to the non-destroyed part of the solution. Thus, when the MIP Insertion is used to repair the solution, one optimizes the MIP model over the variables of the replicas in the sets of unplaced active and passive replicas. If the Related Service Removal is used for destroying the solution prior to calling the MIP Insertion, the latter also optimizes over the replication pattern variables, y_{ir}, of the displaced services.

For the hybrid cloud MIP model of Section 2, the objective function of the MIP Insertion is changed to (32) by adding a second term considering the shared resources reserved for activation of passive replicas, m_{ng}. This is done to ensure that the repair operator inserts passive replicas wisely, even if the insertion does not affect the public cloud cost. However, the public cloud cost term is given a large weight, Ω_C, such that it dominates the expression.

$$\min z = \Omega_C \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{iq} t_{iq} + \sum_{n \in \mathcal{N}(\sigma)} \sum_{g \in \mathcal{G}} m_{ng} \tag{32}$$

The MIP Insertion used in the ALNS for the private cloud model has to consider that the resource constraints of the nodes are relaxed. Therefore, the original node resource constraints (18) are rewritten as (34), where e_{ng} accounts for the resource usage surplus at the nodes, and defined in (35). Similar to the overall objective of the ALNS, the MIP model seeks to minimize the total resource usage surplus of the nodes, hence the objective of the MIP Insertion is written as (33).

$$\min z = \sum_{n \in \mathcal{N}(\sigma)} \sum_{g \in \mathcal{G}} e_{ng} \tag{33}$$

$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} G^A_{iqg} w_{iqn} + \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} G^P_{iqg} v_{iqn} + m_{ng} - e_{ng} \leq 1 \quad \forall n \in \mathcal{N}(\sigma), \forall g \in \mathcal{G} \tag{34}$$

$$e_{ng} \geq 0 \quad \forall n \in \mathcal{N}(\sigma), \forall g \in \mathcal{G} \tag{35}$$

4.4. Local Search Operators

Two classes of local search operators are used in the ALNS: a swap of H_1 replicas from node n_1 with H_2 replicas from node n_2, denoted as the inter-node swap; and a swap of H_1 replicas from node n with H_2 replicas currently placed in the public cloud, denoted as the inter-cloud swap. By using different values for the operator parameters H_1 and H_2, several different swap operators are considered by the ALNS.

4.4.1. Inter-node Swap

The Inter-node Swap operator either swaps H_1 active replicas and H_2 active replicas between two nodes, or swaps two sets of H passive replicas between two nodes. An inter-node swap is defined by the tuple $(\bar{Q}_1, \bar{Q}_2, n_1, n_2)$, where $\bar{Q}_1$ and $\bar{Q}_2$ are sets of replicas and n_1 and n_2 are the nodes where the replicas $\bar{Q}_1$ and $\bar{Q}_2$, respectively, currently are placed. After performing the swap, the replicas in $\bar{Q}_2$ run on n_1, while the replicas in $\bar{Q}_1$ run on n_2.

The operator iteratively searches for swaps that directly or indirectly improve the solution quality. In each iteration, the operator performs an improving swap, and then continues the search. The improvement of a swap is measured by a cost function, ψ. Searching through all feasible swaps in each iteration would make the operator computationally expensive. Therefore, in each iteration, the operator stops the search if an improving swap is found after having searched through all feasible swaps between a given node and any other node. At the end of each iteration, the operator performs the most improving swap found in this iteration. Algorithm 5 outlines the operation of the inter-node swap operator. In Line 8, all swaps between nodes n_1 and n_2 that lead to a feasible solution are evaluated by a cost function, and in Line 10, the most improving swap is saved. When $H_1 = H_2$, we exploit the symmetry by letting n_2 iterate over the nodes that are ordered after n_1 in a fixed ordering (see Line 7). However, when $H_1 \neq H_2$, we also need to consider swaps of H_2 replicas from lower-ordered nodes with H_1 from higher-ordered nodes.

Algorithm 5 Pseudocode of the inter-node swap operator

```
Require: a solution $\sigma$, $H_1 \in \mathbb{N}$, $H_2 \in \mathbb{N}$
 1: //let $(\hat{Q}_1, \hat{Q}_2, n_1, n_2)$ denote the most improving swap in an iteration, and let $\hat{\psi}$ denote the minimum cost
 2: let the set $\tilde{\mathcal{N}}$ contain all nodes in the solution $\sigma$, $\mathcal{N}(\sigma)$
 3: while true do
 4:     $\hat{\psi} = 0$ // set the minimum cost to zero
 5:     while $\tilde{\mathcal{N}} \neq \emptyset$ do
 6:         remove a random element $n_1$ from $\tilde{\mathcal{N}}$
 7:         for all $n_2 \in \mathcal{N}(\sigma)$ : $n_2$ ordered higher than $n_1$ do
 8:             find the most improving swap $(\bar{Q}_1, \bar{Q}_2, n_1, n_2)$ of $H_1$ replicas from $n_1$ with $H_2$ replicas from $n_2$
 9:             if $\psi(\bar{Q}_1, \bar{Q}_2, n_1, n_2) < \hat{\psi}$ then
10:                 $\hat{\psi} = \psi(\bar{Q}_1, \bar{Q}_2, n_1, n_2)$ and $(\hat{Q}_1, \hat{Q}_2, n_1, n_2) = (\bar{Q}_1, \bar{Q}_2, n_1, n_2)$
11:             end if
12:         end for
13:         if $\hat{\psi} < 0$ then break //improving swap found
14:     end while
15:     if $\hat{\psi} = 0$ then break //no improving swap found
16:     perform swap $(\hat{Q}_1, \hat{Q}_2, n_1, n_2)$ //update $n_1, n_2 \in \mathcal{N}(\sigma)$
17:     add nodes $n_1$ and $n_2$ to $\tilde{\mathcal{N}}$ //update $n_2$ if $n_2$ already exists in $\tilde{\mathcal{N}}$
18: end while
19: return $\sigma$
```

We use different cost functions based on whether the operator swaps active or passive replicas, and on whether the ALNS considers the hybrid or the private cloud model. In the ALNS for the hybrid cloud model, we only consider inter-node swaps of passive replicas, and use the cost function ψ_P defined in (36). The functions $\Delta m_g(\bar{Q}_1, \bar{Q}_2, n_1)$ account for the increase or decrease in the shared backup resources reserved for passive replicas (of resource type g) at node n_1, when the set $\bar{Q}_1$ of passive replicas are removed from the node and the set $\bar{Q}_2$ are placed on the node. Thus, ψ_P in (36) computes the total change in the resources reserved for passive replicas. In the ALNS for the private cloud model, it is more natural to look at the increase or decrease in over-utilization of the resources incurred by the swap. Thus, the cost functions measuring the improvement of a swap is simply the difference in the cost function of the ALNS, given in (20), before and after the swap.

$$\psi_P(\bar{Q}_1, \bar{Q}_2, n_1, n_2) = \sum_{g \in \mathcal{G}} \Big(\Delta m_g(\bar{Q}_1, \bar{Q}_2, n_1) + \Delta m_g(\bar{Q}_2, \bar{Q}_1, n_2) \Big) \tag{36}$$

4.4.2. *Inter-cloud Swap*

The behavior of the Inter-cloud Swap operator is similar to the Inter-node Swap operator, but it considers swaps of replicas between a node in the private and the public cloud, instead of swaps between two nodes. Therefore, this operator is solely used in the ALNS for the hybrid cloud model. Moreover, since only active replicas run in the public cloud, this operator considers exclusively swaps of H_1 active replicas with H_2 active replicas. Like the Inter-node Swap, the Inter-cloud Swap tries to limit the search in each iteration by stopping after all feasible swaps between a given node n and the public cloud have been evaluated, as long as one or more improving swaps are found, and then performs the most improving swap. The cost function ψ_A for evaluating the quality of a swap is defined in (37). The function computes the change in public cloud cost when a set $\bar{Q}_1$ of active replicas at a node n is swapped with a set $\bar{Q}_2$ of active replicas currently placed in the public cloud. An improving swap is identified by a negative cost.

$$\psi_A(\bar{Q}_1, \bar{Q}_2) = \sum_{(i,q)\in\bar{Q}_1} C_{iq} - \sum_{(i,q)\in\bar{Q}_2} C_{iq} \tag{37}$$

4.5. *Adaptive Operator Selection*

In the previous sections, we have defined several destroy operators, repair operators and local search operators. In a given iteration of the ALNS, we only use one operator of each type, and we need a way to decide which operators to use in each iteration (see Line 3 of Algorithm 1). The traditional way of selecting the operators in an ALNS is to select the destroy and repair operators independently by using a roulette wheel selection strategy where each of the operators is assigned weights.

In applications where the repair operators are similar in complexity and time consumption, independent selection of the operators seems to be unproblematic. However, in our ALNS where a relatively expensive repair operator, i.e., the MIP insertion, and other less accurate, but faster, repair operators are used side by side, there might be something to gain by coupling the selection of destroy and repair operators. We first select the destroy operator based on the traditional method in ALNS, i.e., a destroy operator $\hat{\theta}$ is selected with a probability according to (38), where Ψ^D_θ represents the weight of the destroy operator $\theta \in O^D$. Moreover, each destroy operator, θ, is associated with a set of repair operators that may be called successively; we denote this set by $O^R_\theta \subseteq O^R$. The idea is to couple the selection of destroy and repair operators, so that some destroy operators have to be followed by a call to the MIP insertion operator, while other operators have to be followed by a less accurate greedy or regret insertion operator. The probability of selecting a repair operator $\hat{\rho} \in O^R_{\hat{\theta}}$ to follow the destroy operator $\hat{\theta}$ is given in (39). At last, the local search operator is chosen in the same way as the destroy operator, that is, operator $\hat{\lambda}$ is selected with a probability according to (40).

$$\frac{\Psi^D_{\hat{\theta}}}{\sum_{\theta\in O^D} \Psi^D_\theta} \tag{38}$$

$$\frac{\Psi^R_{\hat{\rho}}}{\sum_{\rho\in O^R_{\hat{\theta}}} \Psi^R_\rho} \tag{39}$$

$$\frac{\Psi^L_{\hat{\lambda}}}{\sum_{\lambda\in O^L} \Psi^L_\lambda} \tag{40}$$

Generally, the underlying adaptiveness of the ALNS is provided by the *adaptive weight adjustment* principle (Ropke and Pisinger, 2006). The intent is that one could include several destroy and repair operators that function well for different problem instances and different problem structures in the sets of operators, and adjust their weights dynamically based on their past performance. In each iteration of the ALNS, the quality of the solution found and the time spent in the iteration, that is essentially the time spent in Line 5 of Algorithm 1, are used to score the operators used in that iteration. The new solution is classified according to whether it is *a new best solution*, *better than the incumbent solution*, *accepted as a new incumbent*, or *not accepted*; based on this, the operators get a *basic* *score* v_1, v_2, v_3, or v_4, where $v_1 > v_2 > v_3 > v_4$. The lowest score, v_4, is defined as 1. Since the different repair operators might use a significantly different amount of time to repair the destroyed solution, we have chosen to consider the time spent in each iteration in the scoring of the operators. Especially, we are interested in reducing the scores in iterations which are time-consuming. Therefore, we compute a *time-normalized score* shown in (41), where $j \in \{1, 2, 3, 4\}$ according to the classification of the new solution, T_η denotes the time spent in the iteration, and $\bar{T}$ refers to the average time spent in an iteration since the beginning of the search. Moreover, the normalization can be controlled by the parameter ω. The outer max-term in the normalization ensures that no operators are given a score worse than v_4, while the inner min-term makes the scoring only penalize long iterations, and not iterations that are shorter than the average. Moreover, if the repair operator is not capable of finding a feasible solution, the operator gets score v_4.

$$\max\{v_4, v_j \min\{1, \bar{T}/(\omega T_\eta\})\} \tag{41}$$

The search is divided into segments that correspond to 100 iterations. At the beginning of each segment, the scores of all operators are set to one, and throughout the segment, the operators accumulate the time-normalized scores from the iterations where they were used. At the end of a segment, the weights of all operators are updated based on their accumulated scores (Line 12 of Algorithm 1). The destroy operator's weights are updated as shown in (42), where ς_θ^D represents the accumulated scores of destroy operator θ, and Φ_θ^D is a count of the number of times operator θ was called in the last segment. The parameter β_R determines how quickly the past performance is reset in the adaptive weight adjustment method. The repair and local search operators are updated in the same manner.

$$\Psi_\theta^D = (1 - \beta_R)\Psi_\theta^D + \beta_R \frac{\varsigma_\theta^D}{\Phi_\theta^D} \tag{42}$$

4.6. Initial Solution

The construction of the initial solution is done in two steps. First, a replication pattern is selected for each service. Then, the Regret Insertion with $k = 3$ is called to insert all active and passive replicas. Several heuristic rules can potentially be used to select the replication patterns. In the implementation, we have selected the replication patterns that minimize the total resource assigned to all the active replicas in each service. Moreover, for the private cloud model, unless a strict natural upper bound on the number of nodes exists, one has to set a bound. Setting this bound too low might result in a situation where the regret insertion cannot find a feasible solution. If this is the case, one can increase the number of nodes and call the regret insertion until a feasible solution is found.

4.7. Summary of the Operators

Several destroy and repair operators were presented in Sections 4.2 and 4.3, respectively. However, not all of them can be or are used in both the ALNS for the hybrid cloud model and the ALNS for the private cloud model. Moreover, the selection of a repair operator is presented as coupled to the selected destroy method. To achieve the coupled selection, we have grouped destroy and repair operators together, in two groups. The groups for both the hybrid cloud and private cloud cases are shown in Table 1. Keep in mind that the ALNS for the hybrid cloud model also calls the Public Cloud Destruction operator in each iteration, except when the Related Service Removal is used. These groups imply that if Related Service Removal is selected as the destroy operator, the MIP Insertion will be selected as the repair operator. In method group (ii), there is the Greedy Insertion operator and one or more Regret Insertion operators with different k. The different values of k used are decided in the tuning of the algorithm, which is discussed in Section 5.1. Moreover, in the private cloud case, we use two instances of the Over-utilized Node Removal, one for the MIP Insertion and one for the other methods. The two instances might consider different degrees of destruction δ. The choice of grouping is partly based on the results of preliminary tests during the development phase and partly based on the design and functioning of the operators. For instance, calling the MIP Insertion after the random node removal one seems to be more likely to end up in the same solution as before the destruction. Also, the Greedy Insertion and Regret Insertion seem to be more efficient when nodes are only partly destroyed.

Table 1: Grouping of destroy and repair operators

Method Group	Hybrid Cloud		Private Cloud	
	Repair Operators	**Destroy Operators**	**Repair Operators**	**Destroy Operators**
(i)	MIP Insertion	Related Service Removal Worst Node Removal	MIP Insertion	Related Service Removal Over-utilized Node Removal
(ii)	Greedy Insertion Regret Insertion	Random Node Removal Passive Replica Removal Active Replica Removal-ϕ_A Active Replica Removal-ϕ_C	Greedy Insertion Regret Insertion	Random Node Removal Passive Replica Removal Active Replica Removal-ϕ_A Over-utilized Node Removal

Regarding the local search operators, the ALNS for the hybrid cloud model uses one or more Inter-node Swap operators for passive replicas, and one or more Inter-cloud Swap operators for active replicas, where the values of H_1 and H_2 vary. The ALNS for the private cloud model uses Inter-node Swap operators for both active and passive replicas with different values of H_1 and H_2. The combinations of H_1 and H_2 used in the swaps are decided in the tuning process, which is described in Section 5.1.

5. Computational Study

The purpose of the computational study is twofold. We present the results of an evaluation of the ALNS with and without local search (LS) operators. In addition, we perform a comparison of the ALNS with the branch and price (B&P) algorithm proposed by Gullhav and Nygreen (2016). Before presenting the results, we give some details on the setup of the experiments.

5.1. Setup of the Experiments and Tuning

The ALNS was implemented in C++, and compiled with GCC 4.8.2 with optimization option `-O3`. We have run all our experiments on a CentOS 5.8 machine with a dual core 3.0 Ghz Intel E5472 Xeon processor and 16 GB of memory. The MIP Insertion operator calls the Xpress-Optimizer version 27.01.02 of the FICO Xpress Optimization Suite 7.8. The MIP solver of Xpress has utilized up to eight threads in the tree search.

We have designed an instance generator which constructs test cases that aim to be as realistic as possible. The test cases used range from 20 to 70 services, and each service is composed of four components on average, which implies that the largest cases contain a total of 280 components. The instance generator is based on ten different dummy services composed of between three and five components each. The different services have a structure reflecting real-world services with different resource requirements for the different components. In short, each component of the ten services is given a resource requirement distribution, and the case generation is based on drawing resource requirements from these distributions using different seeds. For each instance size, we have generated five cases for testing, and for the instances with 40 and 50 services, we have generated another five for tuning purposes. In all cases, $D = 3$ and $E = 4$, and we consider CPU resources only. The minimum, average, and maximum values on the G^A_{iqg} parameters are 8.0%, 23.0%, and 45.0%, respectively, while the same values for the G^P_{iqg} parameters are 0.33%, 1.6%, and 3.0%. Over all cases, the average number of replication patterns per service is 7.5.

A difference between the test cases for the private cloud model and hybrid cloud model is that the latter have an (effective) upper bound on the number of nodes in the private cloud, N. This upper bound is based on a lower bound (LB) on the number of nodes needed to place all services in the private cloud. This lower bound is computed as the best bound provided by the B&P algorithm (Gullhav and Nygreen, 2016). Using this LB, we constructed two types of test cases for the hybrid cloud: one with $N = \lceil 0.75\text{LB} \rceil$, and one with $N = \lceil 0.9\text{LB} \rceil$. We say that these two types of test cases have 75% and 90% *private cloud coverage*. Furthermore, the placement of active replicas in the public cloud is done at a cost, and the offered VM types used are presented in Table 2. In the table, we list two providers. Since we model the public cloud as a generic pool of resources, we select the VM types for the different components as the cheapest one with enough capacity. The costs are synthetic and not given units. However, the relative cost and size between the VM types reflect the real world.

Table 2: Data of the public cloud VM types used in the hybrid cloud experiments. The capacity is given in percentage of the private cloud node capacity.

Provider 1		Provider 2	
Cost	Capacity	Cost	Capacity
10	10%	15	15%
20	20%	30	30%
40	40%	60	60%

The test cases are named based on the number of services they contain. The hybrid cloud test cases with 20 services are referred to as `H20`, while the corresponding test cases with no restriction on the number of nodes are labeled `P20` and, hence, used for testing the private cloud model. Whenever it is necessary to identify the five different test cases with a given number of services, the cases are appended a letter from 'a' to 'e', i.e., `H20-a` to `H20-e`.

The ALNS has several parameters that can be tuned for the problem structure. We used SMAC (Hutter et al., 2011) to tune the parameters listed in Table 3. The tuning was done on separate test cases with 40 and 50 services, and we set the parameters of the ALNS for the hybrid cloud model and the ALNS for the private cloud model independently. Thus, for the hybrid cloud model we tuned the parameters over 20 cases: ten with 40 services and ten with 50 services. For each size, five of the cases had a private cloud coverage of 75%, while the other five had a private cloud coverage of 90%. The tuning was set up to evaluate the relative gap between the best LB produced by the B&P on the respective test case and the objective value of ALNS after 900 seconds. Moreover, SMAC was run with eight processes in parallel. The processes shared data during the run, but outputted eight different parameter combinations.

Most of the parameters in Table 3 are defined in Section 4. However, the parameters of the acceptance criteria, τ_0 and α, are not tuned directly. These two parameters are set based on the tuned parameters τ_C and P_E, in addition to the maximum number of iterations I and the initial solution σ_0 of a given run. In the ALNS for the hybrid cloud model, the initial temperature τ_0 is set so that a solution with τ_C% higher cost than σ_0 would be accepted with 50% probability. The cooling rate α is set so that after cooling the temperature I times, a solution with τ_C% higher cost than σ_0 would be accepted with probability P_E. For the private cloud model, the temperature and cooling rate is reset in every iteration following a new best solution, since a new best solution has zero cost, i.e., no over-utilization. The reset procedure is similar to the procedure setting the initial temperature and cooling rate for the hybrid cloud model, but when called, it uses the cost of the current solution and the remaining number of iterations.

Table 3: Overview of the tuned parameters and their respective values in the algorithm for the private cloud model (PCM) and hybrid cloud model (HCM).

			Value in	
Parameter	Description	Parameter domain	PCM	HCM
δ^M	Deg. of destruction before calling MIP Insertion	$\{0.05, 0.10, 0.15, \dots, 0.5\}$	0.05	0.05
δ	Deg. of destruction before calling other repair operators	$\{0.05, 0.10, 0.15, \dots, 0.5\}$	0.05	0.1
δ_C^M	Deg. of destr. (public cloud) before calling MIP Insertion	$\{0.1, 0.2, \dots, 1\}$	N/A	0.5
δ_C	Deg. of destr. (public cloud) before calling other repair ops.	$\{0.1, 0.2, \dots, 1\}$	N/A	1.0
γ	Randomness in destroy operators	$\{2, 3, \dots, 8\}$	3	4
β_W	Weight parameter in the Worst Node Removal	$\{0.1, 0.2, \dots, 0.9\}$	0.4	0.1
K	Maximum value of k in the Regret Insertion operators	$\{2, 3, 4, 5\}$	4	4
(ν_1, ν_2, ν_3)	Basic scores in the adaptive operator selection	$\{1, 2, \dots, 50\}$	(37,32,9)	(42,31,22)
ω	Time normalization parameter in the adaptive op. selection	$\{0.5, 0.6, \dots, 1.5\}$	0.6	1.1
β_R	Weight put on the past performance in the scoring	$\{0.05, 0.10, 0.15, \dots, 0.5\}$	0.15	0.5
τ_C	Parameter controlling the temperature initialization	$\{0.1, 0.5, 1, 2.5, 5, 10, 20\}$	1	0.1
P_E	Prob. (in %) of accepting a sol. τ_C% worse than the initial sol.	$\{0.01, 0.05, 0.1, 0.25, 0.5\}$	0.1	0.1

Regarding the tuning of the Regret Insertion operators, Table 3 shows that the maximum k of the operators is 4. This means that we use three Regret Insertion operators with k equal to 2, 3 and 4. The tuning of the swap operators is not shown in the table. However, for both the hybrid cloud and private cloud model, the best combination of Inter-node Swaps of passive replicas were $(H_1, H_2) \in \{(1,1), (2,2)\}$, i.e., swap of one replica with another and swap of two replicas with two other replicas. For the Inter-cloud Swap of active replicas in the hybrid cloud case, the best swap types were $(H_1, H_2) \in \{(1,1),(1,2),(2,1),(2,2)\}$. In the private cloud case, the best Inter-node Swaps of active replicas were $(H_1, H_2) \in \{(1,1),(1,2),(2,2)\}$. In this case, the swaps $(1,2)$ and $(2,1)$ are identical.

All results presented below are obtained by a single run of each test case. When average values for an instance size are presented, the averages are computed based on a single run of each of the five cases of the instance size.

5.2. Added Value of Local Search

First, we are interested in evaluating the effect of using an LS operator on top of the repair operators in the ALNS. In the evaluation, we use the best LB obtained by the B&P algorithm to compute relative gaps between the LB and the objective value of the best solution obtained by the ALNS with and without LS operators. The LBs for the private cloud cases have been shown to be very tight in the cases that are solved to optimality. The private cloud model has structural similarities with the cutting stock problem, for which column generation provides very tight bounds (Vanderbeck, 1999). However, we cannot be certain about the quality of the LBs in the hybrid cloud cases. Table 4 gives the average relative gaps at different points in time, after 5, 10 and 15 minutes, for the hybrid cloud cases with 75% and 90% private cloud coverage. Since the stopping criterion of the ALNS is given by a maximum number of iterations, we have set the maximum number of iterations so that each test case is run slightly longer than 15 minutes. This is done to allow the same computational time independent of the instance size. The average gaps are then computed based on the objective function values at the different points in time. We see that on the smaller cases, the performance of the two versions of the ALNS is quite similar. However, when the problem size grows, there seems to be a benefit in using the LS on top of the repair operators. In addition, the benefit is more pronounced for the cases with 90% private cloud coverage, than for the cases with 75% private cloud coverage. As previous studies have shown, the cases with higher private cloud coverage are more difficult to solve and have larger gaps (Gullhav and Nygreen, 2015, 2016). This might imply that the LS becomes more valuable as the difficulty of the problem increases. Table 5 displays the gaps for the private cloud cases at 5, 10 and 15 minutes of run time. We can see that the relative gaps are smaller than in the hybrid cloud cases, but the ALNS with the LS operators still gives the best results. Furthermore, we also see that when the problem size grows, the LS becomes more beneficial.

Table 4: Average relative gap (in %) between best solution found and best bound at different points in time (seconds): comparison of the ALNS with and without local search (LS) operators on the hybrid cloud cases.

Case	**75% private cloud coverage**						**90% private cloud coverage**					
	ALNS w/o LS			**ALNS with LS**			**ALNS w/o LS**			**ALNS with LS**		
	300	600	900	300	600	900	300	600	900	300	600	900
H20	7.940	6.672	5.938	8.001	6.448	5.639	20.62	18.03	16.42	20.47	17.53	16.41
H30	10.79	8.998	8.013	10.02	8.632	7.859	28.35	24.69	22.41	26.32	22.24	21.07
H40	12.75	11.11	9.781	11.75	10.26	9.397	36.23	29.58	26.92	27.97	24.24	22.49
H50	17.33	14.59	13.27	15.82	12.95	12.17	45.09	36.99	33.53	38.05	31.73	29.20
H60	19.12	15.91	14.63	17.63	14.56	13.52	55.93	43.37	38.11	47.62	37.40	32.86
H70	22.68	19.64	17.29	21.47	17.93	15.66	60.51	49.82	44.31	55.53	42.92	37.26

While Tables 4 and 5 present average values for each instance size, Table 6 presents the p-values of the Wilcoxon signed-rank test (Hollander et al., 2013). This non-parametric statistical hypothesis test is used to compare the relative gaps of the ALNS with and without the LS operators, with the aim to reject the null hypothesis. The null hypothesis states that the difference between the algorithms follows a symmetric distribution around zero, and we test this against the two-sided alternative hypothesis. Moreover, when performing the test, we have grouped the H20 and H30 cases, the H40 and H50 cases, and the H60 and H70 cases together (analogously for the private cloud cases) to obtain 10 observations in each comparison. With a significance level of 5%, we can reject the null hypothesis in all cases in Table 6 where the p-value is less than 0.05. The table show that we cannot reject the null hypothesis with a significance level of 5% for the H20 and H30 cases, except after 300 s with 90% private cloud coverage. However, when the instance size increases, the differences between the two algorithms are significant and in favor of the ALNS with the LS operators. For the private cloud cases, we can see that the difference between the

Table 5: Average relative gap (in %) between best solution found and best bound at different points in time (seconds): comparison of the ALNS with and without local search (LS) operators on the private cloud cases.

Case	ALNS w/o LS			ALNS with LS		
	300	600	900	300	600	900
P20	2.102	2.102	2.102	2.111	1.798	1.528
P30	3.113	2.339	2.339	2.339	1.953	1.953
P40	4.395	3.662	3.074	2.784	2.341	2.341
P50	5.913	5.085	4.260	3.551	3.197	2.959
P60	6.932	5.841	5.349	4.260	3.468	3.266
P70	8.142	6.954	6.277	5.092	4.243	3.987

algorithms is significant at a level of 1%, also for the smaller cases. These results are consistent with the averages presented in Table 4 and 5.

Table 6: P-values of Wilcoxon signed-rank test: comparison of the relative gaps of the ALNS with and without local search operators at different points in time (seconds).

Case	Private cloud coverage	300	600	900
H20-H30		0.813	0.234	0.624
H40-H50	75%	0.049	0.004	0.004
H60-H70		0.064	0.010	0.013
H20-H30		0.014	0.084	0.234
H40-H50	90%	0.006	0.002	0.002
H60-H70		0.002	0.002	0.002
P20-P30		0.002	0.009	0.002
P40-P50	N/A	0.002	0.002	0.002
P60-P70		0.002	0.006	0.002

5.3. Comparison with Exact Approach

When comparing the ALNS with the previously developed B&P algorithm, we need to perform the comparison on a different time scale. Except for the small cases, the B&P use more than 15 minutes to obtain a first feasible integer solution. However, in situations where the service demand has sufficiently long periods of stationarity, one can spend more than 15 minutes optimizing the service deployment, and therefore it is interesting to see the performance of the ALNS heuristic in comparison with the exact B&P on a longer time scale. We are now only presenting a comparison between the B&P and the ALNS with the LS operators, but the results of our experiments show that the ALNS with the LS operators still produces better solutions than the ALNS without the LS. In the following comparisons, we need to underline that the B&P algorithm is designed with the main focus on finding solutions of high quality, not on finding solutions quickly. Therefore, the root node is solved to LP optimality, before a MIP solver is called to optimize over the columns found up to this point, and then branching is performed. To reduce the time to obtain the first integer solution, it would be possible to call a MIP solver before the root node is finished. Nevertheless, this was not considered in the design of the B&P.

Table 7 presents the comparison on the hybrid cloud cases with 75% and 90% private cloud coverage for run times up to three hours. Like the experiments with a shorter run time, the maximum number of iterations for each instance size is set such that each test case is run slightly more than three hours. For the cases with 50 services or less and with 75% private cloud coverage, the B&P algorithm performs better than the ALNS on a long time scale. However, in all of the cases with 50 services or more, the B&P spends longer than an hour to solve the root node of the B&B tree. For all H70 cases, the B&P spends over three hours solving the root node, and does not find a solution within the maximum run time. Moreover, it manages to find a solution in only two of the five H60 cases within three hours. With a private cloud coverage of 90%, the right part of Table 7 shows that the ALNS performs better than the B&P in all cases with 30 services or more. Still, the B&P spends longer than three hours to solve the root node in all of the `H70` cases. Nevertheless, it finds a solution in one of the `H50` cases within one hour, in two of the `H60` cases within two hours, and another two of the `H60` cases within three hours.

Table 7: Average relative gap (in %) between best solution found and best bound at different points in time (seconds): comparison of the B&P and the ALNS with local search (LS) operators on the hybrid cloud cases.

Case	**75% private cloud coverage**								**90% private cloud coverage**							
	B&P				**ALNS with LS**				**B&P**				**ALNS with LS**			
	1800	3600	7200	10800	1800	3600	7200	10800	1800	3600	7200	10800	1800	3600	7200	10800
`H20`	2.875	2.007	1.769	1.709	5.066	4.579	3.765	3.380	11.16	8.044	6.775	5.607	14.02	11.93	10.81	10.01
`H30`	5.349	4.508	3.933	3.854	6.933	5.718	5.030	4.385	20.09	17.74	14.85	12.08	18.28	16.04	13.42	11.63
`H40`	N/A	6.733	5.902	5.489	7.942	6.832	6.246	5.270	24.24*	23.44	18.39	17.64	20.28	17.85	14.87	13.36
`H50`	N/A	N/A	7.663	6.738	10.33	9.247	8.444	7.588	N/A	25.74*	22.45	22.39	25.12	21.90	18.79	17.05
`H60`	N/A	N/A	N/A	8.151*	11.78	10.53	9.156	8.379	N/A	N/A	32.53*	28.04*	27.41	24.17	20.51	18.58
`H70`	N/A	N/A	N/A	N/A	13.85	12.14	10.96	10.28	N/A	N/A	N/A	N/A	30.33	26.26	22.73	20.87

* The algorithm has found the first integer solution in only some of cases within the specified amount of time

Table 8 presents a comparison between the B&P and the ALNS with the LS operators on the private cloud cases. Except for the `P20` cases, the ALNS produces the best solutions after one, two and three hours of run time. For the `P20` cases, the B&P beats the ALNS when given three hours. While none of the solution approaches managed to solve any of the hybrid cloud cases to optimality, the B&P finds and proves the optimal solution in three of the `P20` cases and two of the P30 cases. In comparison, the ALNS manages to find the optimal solution in two of the `P20` cases. Even though the private cloud cases are easier, in terms of lower gaps and more cases solved to optimality, the B&P uses more than an hour to find its first solution in all of the `P70` cases. If one compares the results in Table 8 with Table 5 for the test cases with 40 services or more, one can observe that the ALNS with LS operators produces, on average, solutions with smaller gaps within five minutes than the B&P within three hours. Thus, the ALNS can be said to be much more scalable than the B&P algorithm.

Table 8: Average relative gap (in %) between best solution found and best bound at different points in time (seconds): comparison of the B&P and the ALNS with local search (LS) operators on the private cloud cases.

Case	**B&P**				**ALNS with LS**			
	1800	3600	7200	10800	1800	3600	7200	10800
`P20`	1.831	1.553	0.924	0.620	0.933	0.933	0.933	0.933
`P30`	3.113	2.522	1.916	1.343	1.362	1.160	1.160	0.977
`P40`	4.377*	3.958	3.081	2.938	2.054	1.760	1.465	1.465
`P50`	N/A	5.144*	4.494	4.137	2.482	2.130	1.777	1.653
`P60`	N/A	5.335	4.546	4.352	2.772	2.479	2.081	1.782
`P70`	N/A	N/A	5.183	5.183	3.395	2.799	2.546	2.204

* The algorithm has found the first integer solution in only some of cases within the specified amount of time

Similar to the statistical tests performed for the ALNS with and without LS operators, we have performed the Wilcoxon signed-rank test in order to compare the B&P and the ALNS with LS operators on a longer time scale. The null hypothesis is still that the differences between the algorithms are symmetrically distributed around zero. The most interesting comparisons are on the cases where the B&P is able to find an integer solution within a given time limit. However, to make the comparison complete, we have assigned a high cost to the cases where the B&P did not manage to find the first integer solution within the specified time. The tests are performed with the same grouping of the test cases as before, and the results are shown in Table 9. With a significance level of 5%, we cannot reject the null hypothesis for the `H20` and `H30` with 90% private cloud coverage at any of the points in time. However, with 75% private cloud coverage, there is a significant difference in favor of the B&P algorithm when the algorithms are run 3600 s or more. For the group `H40-H50`, the difference is significant at 5% and in favor of the

ALNS for all run times, except for after 2 hours when the private cloud coverage is 75%. Considering the private cloud cases, the difference is significant in most cases, even at a level of 1% for the larger cases. However, after two and three hours of run time for the group P20-P30, the difference is not significant.

Table 9: P-values of Wilcoxon signed-rank test: comparison of the relative gaps of the B&P and the ALNS with local search operators at different points in time (seconds).

Case	Private cloud coverage	1800	3600	7200	10800
H20-H30		0.084	0.006	0.002	0.004
H40-H50	75%	0.002	0.044	0.193	0.432
H60-H70		0.002	0.002	0.002	0.006
H20-H30		0.322	0.557	0.322	0.160
H40-H50	90%	0.002	0.002	0.006	0.002
H60-H70		0.002	0.002	0.002	0.002
P20-P30		0.014	0.023	0.419	1
P40-P50	N/A	0.002	0.002	0.006	0.002
P60-P70		0.002	0.002	0.006	0.002

To illustrate the evolution of the objective function values in the search process, Figure 2 compares the progress of these values of the B&P and the two ALNS versions on the H40-a case with 75% private cloud coverage. We can see that the objective function value of the ALNS with LS operators drops quicker than the ALNS without LS operators. This effect is observed in almost all cases, and can be explained by the intensifying effect of the LS operators. Moreover, the figure shows that B&P finds a solution after about 40 minutes, and this solution is, in this case, better than the best solution found by the ALNS. While the objective function value of the ALNS algorithm drops gradually, and in small steps, this value drops only two times for the B&P. The B&P finds its best solutions by regularly solving an IP over all columns found up to certain points in time, and this explains the few and distinct drops in the objective function value. Similarly, Figure 3 shows the progress of the cost on the P50-d case. We still see that the ALNS with LS operators drops faster in the beginning of the search, compared to the ALNS without LS operators. The ALNS algorithm produces solutions of higher quality than the B&P in this case, even after three hours. Furthermore, we also observe that the objective function value of the ALNS algorithm now drops fewer times compared to the values in Figure 2. Since the objective function corresponds to the number of nodes required in the solution, one should expect to obtain larger plateaus in the evolution of the cost and, typically, the time spent on a given level increases as the objective function value approaches the optimal solution.

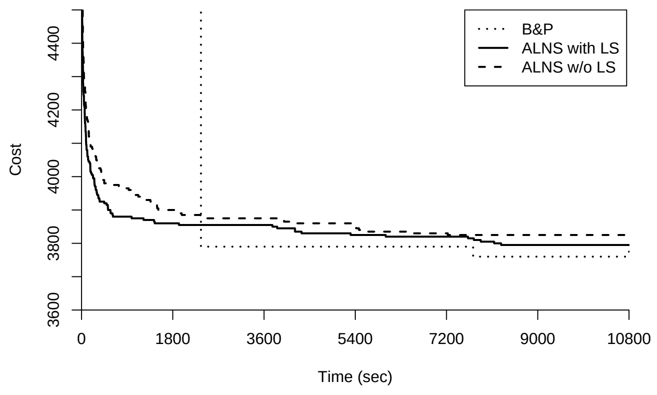

Figure 2: Evolution of objective function value: comparison of the B&P algorithm and the ALNS with and without local search (LS) operators on the H40-a case with 75% private cloud coverage

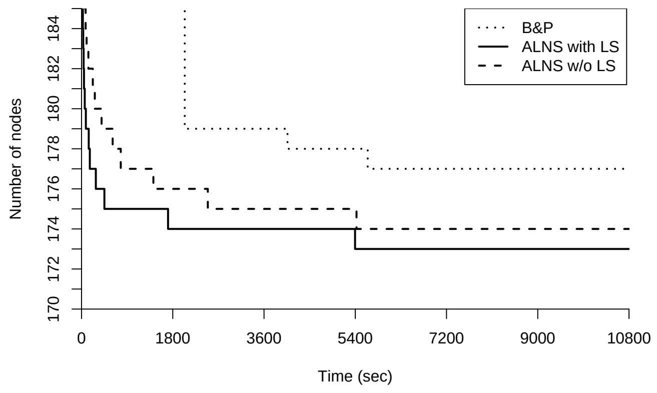

Figure 3: Evolution of objective function value: comparison of the B&P algorithm and the ALNS with and without local search (LS) operators on the `P50-d` case

5.4. Operator Selection Details

In Appendix B, we present detailed results showing the call frequency and average run time of each operator. Since the adaptive operator selection biases the selection based on the operator performance and time consumption (cf. Section 4.5), these results should give some insights into the relative performance of the operators. Here, we extract and discuss some of these results.

Regarding the call frequency of the operators for the hybrid cloud cases, most of the operators have a fairly equal call frequency on all test cases, and are unaffected by the instance size. Two clear exceptions from this are the Passive Replica Removal and the MIP Insertion operators. The Passive Replica Removal is called in about 25% of the iterations in the test cases with 20 and 30 services and 75% private cloud coverage. However, its call frequency is gradually reduced as the instance size increases, and it is called in only about 5% of the iterations in the test cases with 70 services and 75% private cloud coverage. This must mean that the operator in cooperation with a repair and local search operator produces fewer high-quality solutions, such as new best solution, when the instance size grows. In the cases with 90% private cloud coverage, we see the same trend, but the operator is a bit more popular for all instance sizes. A possible explanation of this might be that efficient packing of the passive replicas is less important when the private cloud coverage is smaller and relatively more active replicas have to be placed in the public cloud. Moreover, this effect might be strengthened when the instance size grows. Conversely, the call frequency of the MIP Insertion increases with the instance size, despite that its run time sees a greater relative increase compared to the other repair operators. In the smallest cases, it is called in 25% - 35% of the iterations, while on the largest cases it is called in 50% of the iterations. This must mean that the performance of the MIP operator is very important for the performance of the ALNS, especially on the larger cases. While all operators see an increase in run time per call, the MIP Insertion, which is the most time consuming repair operator in any case, sees the greatest increase, from about 70 milliseconds per call in the smallest cases to almost 2 seconds per call in the largest cases.

For the private cloud cases, the call frequency of Private Replica Removal and MIP Insertion seems to be less affected by the instance size. However, we still see an increase in the average run time per call to the operators as the instance size grows.

To assess the value of the adaptive operator selection, we have run additional experiments without adaptiveness, i.e., where each repair and local search operator are given equal probability of being chosen and these probabilities are not updated during the runs. It turns out that the quality of the best found solutions of the algorithm with adaptiveness in the operator selection is not that much better than the quality of the best found solutions of the algorithm without adaptiveness. However, the average performance of the repair and local search operators seems to be better when using adaptive operator selection, and the average run time of the MIP Insertion is higher without adaptive operator selection. These results, in addition to the benefit of not having to predetermine the weights of the operators, make us believe that the adaptiveness in the operator selection is valuable to the ALNS. Moreover, the adaptiveness should make the ALNS more robust and stable on different families of problem instances. In Appendix B, we show a couple of examples on how the probabilities of selecting the different repair operators evolve during a run of the ALNS.

6. Conclusions

In this paper, we have presented a novel ALNS for the service deployment problem proposed by Gullhav and Nygreen (2015). The ALNS implements a LS layer on top of the repair operators, and the different LS operators are selected dynamically based on their past performance. Furthermore, the ALNS includes a MIP-based repair operator, in addition to faster heuristic insertion operators. Since the MIP Insertion is slow, but produces solutions of better quality compared to the fast heuristic insertion operators, it is necessary to take the time consumption into account in the operator scoring mechanism.

The results of our experiments show that the ALNS benefits from the LS operators on the larger hybrid cloud cases and on all private cloud cases. The results also show that the impact of the LS operators are especially prominent in the first minutes of the search, which can be explained by the operators' intensifying effect.

On a longer time scale, we see that the ALNS with LS operators performs significantly better than a previously proposed B&P algorithm on the larger test cases. However, on the smaller cases, the differences between the algorithms are not significant or in favor of the B&P. On the larger private cloud cases, the ALNS with LS operators produces after five minutes of run time as good solutions as the B&P manages to produce after three hours.

Acknowledgments

We are grateful to the reviewers for their comments which have helped us to improve the quality of this paper.

Appendix A. Mathematical Symbols

Table A.1 summarizes the main mathematical symbols used in the formulations of Section 3 and the description of the ALNS in Section 4. The symbols are ordered alphabetically with latin letters ordered before greek letters.

Appendix B. Detailed Results of the Operator Selection

Tables B.2 - B.5 give some details on the operator selection frequencies and the average run time of each repair and local search operators for the different cases. Each result is an average over the results of the five cases of the same instance size. Due to space issues, the names of the operators are abbreviated according to Table B.1.

As discussed in Section 5.4, running the ALNS without adaptive operator selection, does not give a significant increase in the quality of the best found solution. However, the average performance of the repair and local search operators is reduced, and the run time of the MIP Insertion operator is increased. Figures B.1 and B.2 show the evolution of the probabilities of selecting the different repair operators, which are due to the adaptive operator selection. Note that these probabilities are updated every 100th iteration. From the figures, one can see the probabilities seem to be constantly fluctuating, and the probability for selecting a given repair operator is quite different at different phases of the search. We interpret this as meaning that the performance of the operators and the run time of the MIP Insertion vary throughout the search, and hence, there is a benefit of using the adaptive operator selection strategy discussed in Section 4.5.

Table A.1: Overview of the main mathematical symbols in the paper in alphabetical order (latin before greek letters)

B_g	Capacity of resource type g at each node
C_{iq}	Cost of deploying an active or passive replica of component q of service i in the public cloud
$c(\sigma)$	Cost of solution σ
D	Upper bound on the number of different services deployed on a node
E	Upper bound on the number of passive replicas deployed on a node
e_{ng}	Variable in the optimization model equaling resource usage surplus of resource g at node n
$\mathcal{G}$	Set of resource types
$G_g(n)$	Total amount of resources of type g assigned to replicas placed on node n, including resources reserved to activation of passive replicas
G^A_{iqg}	The amount of resources of type g guaranteed to an active replica of component q of service i
G^P_{iqg}	The amount of resources of type g guaranteed to a passive replica of component q of service i
H_X	Number of replicas to swap from node 1 to node 2 ($X = 1$) and from node 2 to node 1 ($X = 2$) in the local search operators
h	Number of nodes to be destroyed (Related Service Removal uses h_N and h_S as the numbers of nodes and services to be destroyed)
I	Maximum number of iterations
k	Parameter of the Regret Insertion operator specifying the shortsightedness
m_{ng}	Variable in the optimization model equaling the amount of resources of type g reserved on node n to allow for activation of a passive replica
$\mathcal{N}$	Set of nodes
$\mathcal{N}(\sigma)$	Set of nodes available for placement for solution σ
O^X	Set of destroy ($X = D$), repair ($X = R$) and local search ($X = L$) operators
$\mathcal{Q}_i$	Set of components of service i
$\mathbf{Q}_C(\sigma)$	Vector of active replicas placed in the public cloud in solution σ
$\mathcal{Q}_X(n)$	Set of active replicas ($X = A$) and passive replicas ($X = P$) placed on node $n \in \mathcal{N}(\sigma)$ of a solution σ
$\mathcal{R}_i$	Set of replication patterns for service i
R^A_{iqr}	The number of active replicas of component q of service i in replication pattern r
R^P_{iqr}	The number of passive replicas of component q of service i in replication pattern r
$\mathcal{S}$	Set of services
s_{in}	Variable in the optimization model equaling 1 if at least one replica of service i is deployed on node n, and 0 otherwise
T_η	The CPU time spent in the iteration η
$\bar{T}$	The average CPU time of an iteration
t_{iq}	Variable in the optimization model equaling the number of active replicas of component q of service i deployed in the public cloud
u_n	Variable in the optimization model equaling 1 if node n is turned on, and 0 otherwise
v_{iqn}	Variable in the optimization model equaling 1 if a passive replica of component q of service i is deployed on node n, and 0 otherwise
w_{iqn}	Variable in the optimization model equaling 1 if an active replica of component q of service i is deployed on node n, and 0 otherwise
y_{ir}	Variable in the optimization model equaling 1 if replication pattern r is chosen for service i, and 0 otherwise
α	Cooling factor of the simulated annealing criterion
β_R	Parameter used to weight the historic operator performance in the adaptive operator selection
β_W	Parameter used to weight low resource utilization by active replicas and low resource utilization by passive replicas in the $\phi_W(n)$ feature
γ	Degree of randomization of the destroy operators
$\Delta e^X_g(i, q, n)$	The increase in over-utilization of resource g when an active replica ($X = A$) or a passive replica ($X = P$) of component q of service i is inserted on node n
δ	Degree of destruction of the destroy operators
$\Delta m_g(\bar{\mathcal{Q}}_1, \bar{\mathcal{Q}}_2, n_1)$	Increase of decrease in the shared backup resources for passive replicas (of resource type g) at node n_1, when the set $\bar{\mathcal{Q}}_1$ of passive replicas are removed from the node and the set $\bar{\mathcal{Q}}_2$ are placed on the node.
$\zeta_{iq}(k)$	The value of inserting an active of passive replicas at its best node in Regret Insertion with parameter k
η	Iteration counter
θ	destroy operator
λ	local search operator
ν_X	Basic score of classification X ($X = 1, 2, 3, 4$)
$\xi_X(i, q, n)$	Cost of inserting an active replica ($X = A$) or a passive replica ($X = P$) of component q of service i on node n
π	Random number
ρ	repair operator
σ	Solution ($\bar{\sigma}$ - incumbent; $\hat{\sigma}$ - best found)
ς^X_χ	The accumulated score of destroy ($X = D$), repair ($X = R$) or local search ($X = L$) operator χ in the last segment
τ	Temperature of the simulated annealing criterion in the current iteration (τ_0 - initial temperature)
Φ^X_χ	The number of times destroy ($X = D$), repair ($X = R$) or local search ($X = L$) operator χ was called in the last segment
$\phi_X(n)$	Function used to bias the node destruction towards a specific feature, such as low resource utilization by active replicas ($X = A$), low resource utilization by passive replicas ($X = P$), low weighted resource utilization by active and passive replicas ($X = W$), low average public cloud cost of active replicas ($X = C$), and large over-utilization ($X = O$)
$\mathbf{\Psi}^X$	Vector of weights of the destroy ($X = D$), repair ($X = R$) and local search ($X = L$) operators
$\psi_X(\cdot)$	Cost of performing a swap of active replicas ($X = A$) and passive replicas ($X = P$)
Ω_X	Large weight put on specific terms in weighted expressions ($X = O, C, I$)
ω	Parameter used to control the penalization of operators due to high CPU time usage

Table B.1: Abbreviations of the ALNS operators

Operator type	**Abbreviation**	**Description**
Destroy operators	ARR-ϕ_A	Active Replica Removal with feature ϕ_A
	ARR-ϕ_C	Active Replica Removal with feature ϕ_C
	ONR-M	Over-utilized Node Removal coupled with MIP Insertion
	ONR	Over-utilized Node Removal not coupled with MIP Insertion
	PRR	Passive Replica Removal
	RNR	Random Node Removal
	RSR	Related Service Removal Insertion
	WNR	Worst Node Removal
Repair operators	GI	Greedy Insertion
	MIP	MIP Insertion
	RI-k	Regret Insertion with parameter k
Local Search Operators	CA-H_1-H_2	Inter-cloud Swap of H_1 active replicas from the private cloud with H_2 active replicas from the public cloud
	NA-H_1-H_2	Inter-node Swap of H_1 active replicas from one node with H_2 active replicas from another node
	NP-H_1-H_2	Inter-node Swap of H_1 passive replicas from one node with H_2 passive replicas from another node

Table B.2: Call frequency of the destroy, repair and local search operators for the hybrid cloud cases with different private cloud coverage (PCC).

Case	**PCC**	**Destroy operators**						**Repair operators**					**Local search operators**					
		RSR	WNR	RNR	PRR	ARR-ϕ_A	ARR-ϕ_C	MIP	GI	RI-2	RI-3	RI-4	NP-1-1	NP-2-2	CA-1-1	CA-1-2	CA-2-1	CA-2-2
H20	75%	0.247	0.128	0.122	0.253	0.131	0.120	0.374	0.118	0.169	0.169	0.169	0.200	0.161	0.160	0.164	0.157	0.157
H30	75%	0.259	0.130	0.122	0.235	0.133	0.121	0.388	0.111	0.166	0.167	0.167	0.202	0.161	0.161	0.163	0.156	0.157
H40	75%	0.336	0.136	0.116	0.150	0.136	0.126	0.471	0.101	0.140	0.143	0.145	0.230	0.165	0.152	0.157	0.149	0.147
H50	75%	0.292	0.147	0.128	0.120	0.161	0.151	0.439	0.112	0.151	0.152	0.146	0.188	0.171	0.163	0.164	0.157	0.157
H60	75%	0.339	0.153	0.124	0.082	0.156	0.146	0.492	0.105	0.135	0.134	0.133	0.231	0.183	0.152	0.150	0.145	0.139
H70	75%	0.364	0.151	0.122	0.056	0.162	0.143	0.515	0.105	0.129	0.127	0.124	0.241	0.192	0.146	0.146	0.140	0.136
H20	90%	0.130	0.104	0.149	0.388	0.130	0.099	0.234	0.152	0.205	0.205	0.204	0.176	0.161	0.166	0.177	0.159	0.161
H30	90%	0.196	0.128	0.138	0.280	0.140	0.118	0.324	0.138	0.180	0.178	0.180	0.187	0.163	0.163	0.170	0.157	0.160
H40	90%	0.235	0.130	0.136	0.240	0.140	0.119	0.365	0.126	0.170	0.171	0.168	0.218	0.165	0.155	0.161	0.148	0.153
H50	90%	0.258	0.139	0.146	0.185	0.147	0.126	0.397	0.119	0.162	0.161	0.161	0.186	0.164	0.162	0.167	0.159	0.162
H60	90%	0.295	0.149	0.141	0.144	0.144	0.127	0.444	0.112	0.147	0.149	0.148	0.207	0.170	0.155	0.164	0.149	0.156
H70	90%	0.348	0.153	0.135	0.106	0.137	0.120	0.501	0.099	0.132	0.133	0.134	0.217	0.179	0.148	0.166	0.141	0.149

Table B.3: Average run time (in seconds) per call of the repair and local search operators for the hybrid cloud cases with different private cloud coverage (PCC).

Case	**PCC**	**Repair operators**					**Local search operators**					
		MIP	GI	RI-2	RI-3	RI-4	NP-1-1	NP-2-2	CA-1-1	CA-1-2	CA-2-1	CA-2-2
H20	75%	0.074	0.002	0.002	0.002	0.002	0.003	0.007	0.002	0.019	0.005	0.027
H30	75%	0.152	0.007	0.007	0.007	0.007	0.006	0.016	0.004	0.064	0.011	0.081
H40	75%	0.352	0.016	0.015	0.015	0.015	0.011	0.029	0.007	0.182	0.019	0.261
H50	75%	0.739	0.026	0.026	0.026	0.026	0.018	0.045	0.010	0.387	0.030	0.744
H60	75%	1.162	0.039	0.040	0.040	0.040	0.028	0.069	0.015	0.632	0.040	0.974
H70	75%	1.720	0.057	0.060	0.060	0.060	0.039	0.097	0.020	1.006	0.056	1.575
H20	90%	0.071	0.003	0.002	0.002	0.002	0.003	0.007	0.001	0.007	0.005	0.012
H30	90%	0.176	0.007	0.007	0.007	0.007	0.006	0.016	0.003	0.020	0.010	0.029
H40	90%	0.448	0.015	0.014	0.014	0.014	0.011	0.029	0.005	0.054	0.018	0.073
H50	90%	0.778	0.029	0.028	0.028	0.027	0.018	0.045	0.008	0.111	0.027	0.170
H60	90%	1.474	0.048	0.047	0.047	0.046	0.027	0.070	0.011	0.149	0.034	0.178
H70	90%	1.944	0.076	0.077	0.076	0.075	0.038	0.096	0.015	0.262	0.045	0.304

Table B.4: Call frequency of the destroy, repair and local search operators for the private cloud cases.

Case	Destroy operators						Repair operators					Local search operators				
	RSR	ONR-M	RNR	ONR	PRR	ARR-ϕ_A	MIP	GI	RI-2	RI-3	RI-4	NP-1-1	NP-2-2	NA-1-1	NA-1-2	NA-2-2
P20	0.109	0.111	0.185	0.166	0.266	0.162	0.220	0.182	0.208	0.197	0.192	0.203	0.195	0.208	0.198	0.195
P30	0.126	0.127	0.159	0.152	0.285	0.152	0.252	0.161	0.203	0.195	0.188	0.205	0.196	0.206	0.198	0.195
P40	0.130	0.130	0.145	0.138	0.307	0.151	0.259	0.149	0.203	0.197	0.191	0.207	0.198	0.203	0.197	0.194
P50	0.131	0.133	0.146	0.136	0.299	0.156	0.264	0.140	0.206	0.199	0.191	0.211	0.197	0.205	0.196	0.190
P60	0.125	0.125	0.142	0.133	0.326	0.150	0.250	0.138	0.213	0.204	0.196	0.208	0.197	0.211	0.196	0.189
P70	0.127	0.126	0.144	0.131	0.319	0.154	0.252	0.130	0.214	0.205	0.199	0.209	0.199	0.208	0.194	0.189

Table B.5: Average run time (in seconds) of the repair and local search operators for the private cloud cases.

Case	Repair operators					Local search operators				
	MIP	GI	RI-2	RI-3	RI-4	SP-1-1	SP-2-2	SA-1-1	SA-1-2	SA-2-2
P20	0.075	0.001	0.001	0.001	0.001	0.003	0.006	0.007	0.012	0.020
P30	0.221	0.002	0.003	0.003	0.003	0.007	0.015	0.014	0.027	0.043
P40	0.526	0.004	0.005	0.005	0.005	0.010	0.023	0.023	0.045	0.073
P50	0.950	0.006	0.008	0.008	0.008	0.015	0.035	0.032	0.067	0.111
P60	1.654	0.010	0.012	0.012	0.012	0.017	0.036	0.036	0.082	0.127
P70	2.104	0.012	0.016	0.016	0.016	0.020	0.041	0.045	0.103	0.157

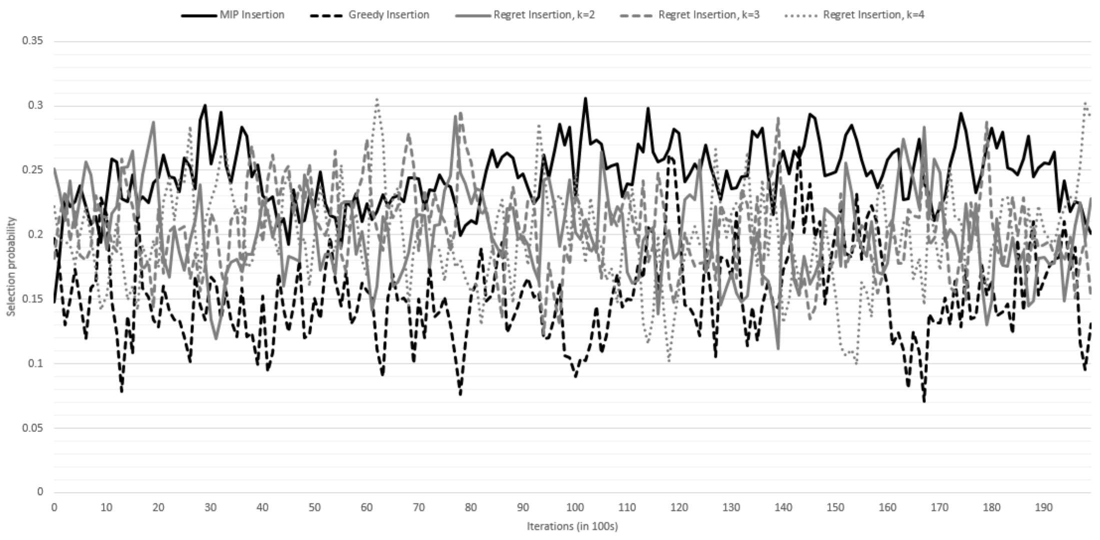

Figure B.1: Evolution of the probabilities of selecting the different repair operators in the `H30-e` case with 75% private cloud coverage.

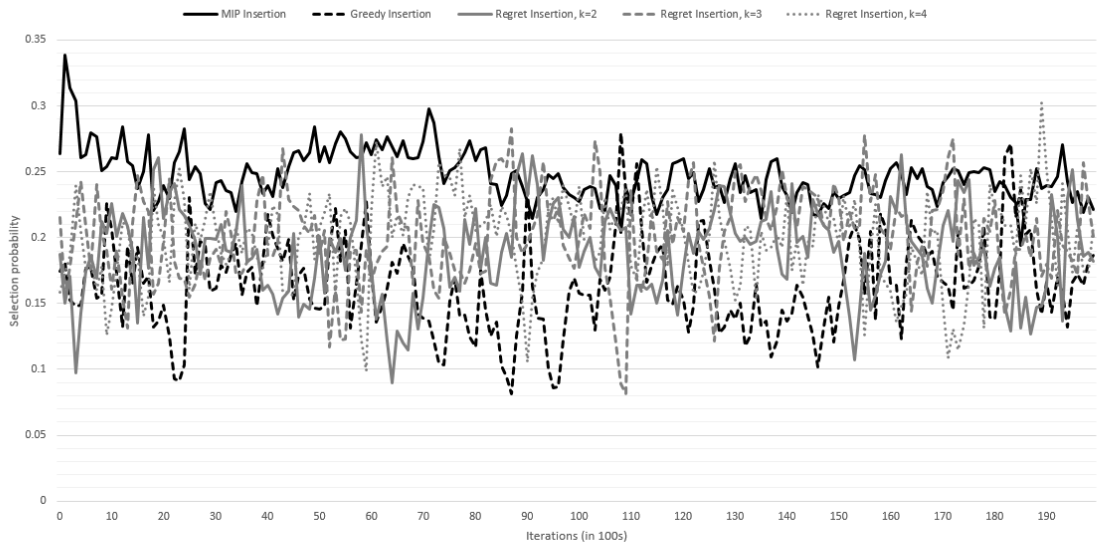

Figure B.2: Evolution of the probabilities of selecting the different repair operators in the `H40-c` case with 90% private cloud coverage.

References

Amazon Web Services. Amazon Web Services (AWS) - Cloud Computing Services, 2015. URL `http://aws.amazon.com/`. Last visited 2015/03/04.

D. Ardagna, B. Panicucci, M. Trubian, and L. Zhang. Energy-aware autonomic resource allocation in multitier virtualized environments. *IEEE Transactions on Services Computing*, 5(1):2–19, 2012.

A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

A. Beloglazov, R. Buyya, Y. C. Lee, and A. Y. Zomaya. A taxonomy and survey of energy-efficient data centers and cloud computing systems. *Advances in Computers*, 82(2):47–111, 2011.

E. Bin, O. Biran, O. Boni, E. Hadad, E. Kolodner, Y. Moatti, and D. Lorenz. Guaranteeing high availability goals for virtual machine placement. In *2011 31st International Conference on Distributed Computing Systems*, pages 700–709, 2011.

B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield. Remus: High availability via asynchronous virtual machine replication. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 161–174, Berkeley, CA, USA, 2008. USENIX.

T. Distler, R. Kapitza, I. Popov, H. P. Reiser, and W. Schröder-Preikschat. SPARE: Replicas on hold. In *Proceedings of the 18th Network and Distributed System Security Symposium*, Geneva, Switzerland, 2011. The Internet Society.

H. Goudarzi and M. Pedram. Multi-dimensional SLA-based resource allocation for multi-tier cloud computing systems. In *2011 IEEE 4th International Conference on Cloud Computing*, pages 324–331, Los Alamitos, CA, USA, 2011. IEEE Computer Society.

A. N. Gullhav and B. Nygreen. Deployment of replicated multi–tier services in cloud data centres. *International Journal of Cloud Computing*, 4(2):130–149, 2015.

A. N. Gullhav and B. Nygreen. A branch and price approach for deployment of multi-tier software services in clouds. *Computers & Operations Research*, 75:12 – 27, 2016.

A. N. Gullhav, B. Nygreen, and P. E. Heegaard. Approximating the response time distribution of fault-tolerant multi-tier cloud services. In *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pages 287–291, Los Alamitos, CA, USA, 2013. IEEE Computer Society.

P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449 – 467, 2001.

M. Hollander, D. A. Wolfe, and E. Chicken. *Nonparametric statistical methods*. John Wiley & Sons, Somerset, NJ, USA, 3rd edition, 2013.

F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In C. A. C. Coello, editor, *Learning and Intelligent Optimization*, volume 6683 of *Lecture Notes in Computer Science*, pages 507–523. Springer Berlin Heidelberg, 2011.

B. Jennings and R. Stadler. Resource management in clouds: Survey and research challenges. *Journal of Network and Systems Management*, 23(3):567–619, 2015.

R. Jhawar, V. Piuri, and M. Santambrogio. Fault tolerance management in cloud computing: A system-level perspective. *IEEE Systems Journal*, 7(2):288–297, 2013.

W. Kuo and R. Wan. Recent advances in optimal reliability allocation. In G. Levitin, editor, *Computational Intelligence in Reliability Engineering*, volume 39 of *Studies in Computational Intelligence*, pages 1–36. Springer Berlin Heidelberg, 2007.

H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search: Framework and applications. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, pages 363–397. Springer, Boston, 2010.

S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, and A. Ghalsasi. Cloud computing — the business perspective. *Decision Support Systems*, 51(1):176 – 189, 2011.

P. Mell and T. Grance. The NIST definition of cloud computing, 2011. NIST SP 800-145.

S. N. Parragh and V. Schmid. Hybrid large neighborhood search for the dial-a-ride problem. In *Proceeding of the VII ALIO/EURO—workshop on applied combinatorial optimization*. ALIO-EURO, 2011.

S. N. Parragh and V. Schmid. Hybrid column generation and large neighborhood search for the dial-a-ride problem. *Computers & Operations Research*, 40(1):490 – 497, 2013.

D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403 – 2435, 2007.

D. Pisinger and S. Ropke. Large neighborhood search. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, pages 399–419. Springer, Boston, 2010.

G. R. Raidl and J. Puchinger. Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization. In C. Blum, M. J. B. Aguilera, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics: An Emerging Approach to Optimization*, pages 31–62. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

S. Ropke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006.

G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt, and G. Dueck. Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, 159(2):139–171, 2000.

P. Shaw. A new local search algorithm providing high quality solutions to vehicle routing problems. Technical report, Department of Computer Science, University of Strathclyde, Glasgow, Scotland, UK, 1997.

F. Vanderbeck. Computational study of a column generation algorithm for bin packing and cutting stock problems. *Mathematical Programming*, 86(3):565–594, 1999.