Journal of AI and Data Mining
Vol 8, No 1, 2020, 55-65.
DOI: 10.22044/JADM.2019.7934.1931

Controller Placement in Software Defined Network using Iterated Local Search

A. Moradi [1*], A. Abdi Seyedkolaei [2], S. A. Hosseini Seno [2]

1. Faculty of Mathematical Sciences, University of Mazandaran, Babolsar, Iran.
2. Faculty of Engineering, Ferdowsi University of Mashhad, Mashhad, Iran.

Received 18 January 2019; Revised 03 September 2019; Accepted 30 November 2019
**Corresponding author: A.moradi@umz.ac.ir (A. Moradi).*

Abstract
The software defined network is a new computer network architecture that separates the controller and data layers in network devices such as switches and routers. By the emergence of software defined networks, a class of location problems, called the controller placement problem, has attracted much more research attention. The task in the problem is to simultaneously find the optimal number and location of controllers satisfying a set of routing and capacity constraints. In this paper, we suggest an effective solution method based on the so-called iterated local search strategy. We then, compare our method with the existing standard mathematical programming solvers on an extensive set of problem instances. It turns out that our suggested method is computationally much more effective and efficient over middle to large instances of the problem.

Keywords: *Software Defined Network, Controller Placement, Iterated Local Search.*

1. Introduction

The architecture of network communication equipment including switches and routers consists of the two planes of control and data [1]. The control plane provides the necessary information for routing in the network. The data plane has the task of transferring packets from the input port to the output port based on the information entering in its routing table. In the recent years, researchers have focused their attention on separating the control plane from the data plane. In this separation, the control plane is located on the server or a program named the controller. The data plane remains in the switch or router. Separation of the control plane from the data plane has many benefits, such as more programmability of the control plane, possibility of network virtualization, reduction of the operating costs, and greater independence of the network equipment manufacturers. The above idea has led to the emergence of software defined networks (SDNs) [2]. The emerge of SDN has attracted the attention of a large number of universities and industries to its implementation in its communication infrastructure [3]. On these networks, configuration methods are often simpler and more precise, and the possibility of using more physical infrastructures is provided [4]. A group of network operators, network service providers, and network equipment manufacturers created an organization called ONF [5] to promote SDN networks and provide a standard communication protocol of network.

In SDN networks, the controller is often responsible for the propagation of each flow in the network, and performs this by assigning input flows to the switches [6]. This work has given a pivotal role to the controller, as with the aim of it, complete knowledge of the network can be applied in optimizing flow management and support of user requirements [7].

One of the important problems defined on SDN networks is the *controller placement problem*. In this problem, the goal is to determine the location of the controllers and their required number in the network so that the total cost of installing controllers, the cost of connecting switches to controllers, and the cost of connecting controllers to each other is minimized [8]. In terms of computational complexity, the problem is NP-hard [9].

In work, an algorithm called *Iterated Local Search* (ILS) [10] is used to tackle the problem. This algorithm uses concepts such as the neighborhood and perturbation mechanism to build an efficient topology of the network at an acceptable running time.

Iterated local search works by relying on a clear principle that is easy to identify and that usually leads to high-performing algorithms. In addition, ILS algorithms are relatively intuitive to design, and at the same time malleable. Besides, ILS often offers an excellent trade-off between simplicity and flexibility. Therefore, iterated local search often serves as a basis for a larger algorithm engineering effort if high-performing heuristics are desired. This approach is very successful, as shown by the various problems for which ILS algorithms have been or still are state of the art [24].

The remainder of this paper is organized as what follows. Section 2 describes the efforts that have been made in the past to solve problems. Section 3 is focused on describing a standard accurate programming formulation of the problem. Section 4 is dedicated to the introduction of the proposed algorithm based on the ILS method and its structure. Section 5 provides computational analysis of the ILS algorithm, and investiges its performance in comparison with standard solvers for the programming formulations. Finally, conclusions are presented in Section 6.

2. Related works

In this section, the works done on the controller placement problem are reviewed. Heller et al. in [8] have used two criteria, namely average latency and worst-case latency, to select an appropriate controller location. In their method, first, the shortest paths between switches are computed, and then the average latency and largest length of such paths are considered as average and worst-case latency, respectively. Xiao et al. in [11] have presented a method for controller placement in the SDN wide network. In this method, first, the spectral clustering algorithm divides the network into several domains, and then the best location to install the controller is specified in each domain. Unlike the two previous works, the research work conducted in [12] addresses the reliability of SDN networks. In this work, the reliability of receiving information from switches in a location will increase the priority of choosing it to install a controller, while other important parameters in the controller placement, such as cost, delay, and load balancing are not considered.

The focus on the properties of network graph has caused Zhang et al. [13] to use a graph parsing algorithm to choose proper locations for controller installation, which minimizes the probability of losing the switch-controller connection. Obadia et al. in [14] have considered the controller placement problem by applying tree topology to connect the controllers to each other. The induction of such a topology to controllers will reduce their operational overhead. Yao et al. in [15] have investigated the controller placement problem in the presence of controllers with a limited capacity. In [16], Zhang and others have considered different topologies for connecting controllers.

Recently, Sallahi et al. in [17], in order to minimize the network cost, have described the controller placement problem as a sum of several objective functions. They reasoned that the controller placement should meet certain constraints, especially for the control plane. Their idea simultaneously determines the optimal number, location, and type of controller, as well as the communications between the network elements, and to limit, the network cost is considered the various constraints. The proposed method is based on solving a mathematical programming formulation that is suitable only for small-scale networks. The mathematical programming model proposed in [17] and its exact solution based on the use of standard solvers of mathematical programming problems will be a comparative criterion in this work.

3. Mathematical formulation of problem

In this section, the mathematical formulation of the controller placement problem, taken from [17], is described. This model includes a set of switches, a set of controllers, a set of links, and a set of possible locations for installing the controllers represented by the symbols S, C, L, and P, respectively. In this formulation, the network is assumed to be a graph, each node of which may locate a switch or a controller. In a special case, a switch and a controller can be located together. In a node, each switch is connected only to one of the controllers. Considering the limit in the number of ports of a controller, several switches can be managed by a controller. In this graph, each controller must be in a direct communication with the other controllers. Hence, the topology used between the controllers is of full mesh type. For each switch s ϵ S, the number of packets sent to the controller is represented by the parameter σ^s. Each controller c ϵ C has the parameters $\alpha^c, \mu^c, \mathrm{K}^c,$ and φ^c which, respectively, represent the number of ports, number of packets, cost, and number of controllers of type *c*. For each link $l \epsilon$ L, the two parameters ω^l and ϕ^l

are defined as to express the bandwidth and the installation cost. The other (non-negative) parameters of the problem are the packet size, network latency, and packet processing time in a controller, which are represented by the symbols β, γ, and δ, respectively. Now, according to the definitions of the sets and parameters, the decision variables of the problem are as follows:

$$x_{cp} = \begin{cases} 1 & \text{If controller } c \text{ is located in position } p; \\ 0 & \text{otherwise} \end{cases}$$

$$v_{sp}^{l} = \begin{cases} 1 & \text{If there is a link } l \text{ between switch } s \text{ and} \\ & \quad \text{the controller located in position } p; \\ 0 & \text{otherwise} \end{cases}$$

$$z_{pq}^{l} = \begin{cases} 1 & \text{If link } l \text{ exists between positions } p \text{ and } q; \\ 0 & \text{otherwise} \end{cases}$$

The goal is then to minimize the network deployment costs. These costs include the cost of installing the controller in locations, the costs of connecting switches to controllers, and the costs of connecting controllers to each other, represented by the symbols $C_c(x), C_l(v)$, and $C_t(z)$, respectively. Equations (1), (2), and (3) describe how each one of these costs is calculated. The *dist(a,b)* function in Equations (2) and (3) obtains the Euclidean distance between location *a* and location *b*.

$$C_C(x) = \sum_{c \in C} K^c \sum_{p \in P} x_{cp} \tag{1}$$

$$C_l(v) = \sum_{l \in L} \Phi^l \sum_{s \in S} \sum_{p \in P} dist(s,p) v_{sp}^{l} \tag{2}$$

$$C_t(z) = \sum_{l \in L} \Phi^l \sum_{\substack{s \in S \\ q < p}} \sum_{p \in P} dist(p,q) z_{pq}^{l} \tag{3}$$

In the following, the objective function of the problem is expressed along with its constraints:

$$Minimize(C_c(x) + C_l(v) + C_t(z)) \tag{4}$$

$$\sum_{c \in C} x_{cp} \le 1, \quad \forall p \in P \tag{5}$$

$$\sum_{q \in P} \sum_{l \in L} (z_{pq}^{l} + z_{qp}^{l}) + \sum_{s \in S} \sum_{l \in L} v_{sp}^{l} \le \sum_{c \in C} \alpha^c x_{cp}, \quad \forall p \in P \tag{6}$$

$$\sum_{l \in L} \sum_{p \in P} v_{sp}^{l} = 1, \quad \forall s \in S \tag{7}$$

$$\sum_{l \in L} \sum_{s \in S} \sigma^s v_{sp}^{1} \le \sum_{c \in C} \mu^c x_{cp}, \quad \forall p \in P \tag{8}$$

$$\sum_{p \in P} x_{cp} \le \varphi^c, \quad \forall c \in C \tag{9}$$

$$g(\sigma^s, \beta) \le \sum_{l \in L} f(w^l) v_{sp}^{l} \quad \forall s \in S, \ \forall p \in P \tag{10}$$

$$2Tran(v) + 2\Pr op(x,v) + 2\Pr oc(x) \le \gamma \tag{11}$$

$$\sum_{c \in C} x_{cq} + \sum_{c \in C} x_{cp} \le \sum_{l \in L} z_{pq}^{l} + 1, \quad (q < p, \forall q \in P, \forall p \in P) \tag{12}$$

$$x_{cp} \in \{0,1\} \quad \forall c \in C, \quad \forall p \in P \tag{13}$$

$$v_{sp}^{l} \in \{0,1\} \quad \forall l \in L, \quad \forall s \in S, \forall p \in P \tag{14}$$

$$z_{pq}^{l} \in \{0,1\} \quad \forall l \in L, \quad \forall p \in P, \forall q \in P \tag{15}$$

The constraints (5) and (6) describe the limitation of the number of controllers of type $c \in C$ in place *p* and the number of ports of a controller. Constraint (7) indicates the limitation of the use of link *l* between a switch *s* and a controller *c*. The constraints (8) and (9) represent the limit of the number of packet processed by controller *c* and the inventory of each controller. Constraint (10) indicates the bandwidth limitation of link *l*. The function $g(a,b)$ in (10), with two input arguments *a* and *b*, obtains the number of *a* packets of length *b* bytes based on bytes per second. *F*(*a*) is a function that converts a value in megabytes or gigabytes (*a*) to bytes per second. The constraints (11) and (12) represent the total network delay, i.e. the sum of transmission delay by switch *s*, propagation delay between the controller *c* and the switch *s*, and processing delay of the controller *c* with the symbols *Tran(v)*, *Prop(x,v)*, *Proc(x)*, respectively, should be less than or equal to the total network delay. Constraint (12) imposes a full mesh topology. Finally, the constraints (13), (14), and (15) represent the binary values that the decision variables can have. Each binary assignment to the problem variables that follow the constraints (5) to (12) is called a *feasible solution* of the problem or simply a *problem solution*, in short**.**

4. Proposed algorithm based on iterated local search

In this section, first, the general structure of the iterated local search algorithm is described, and then the types of neighborhood structures and proposed perturbation mechanisms are expressed in the algorithm. In mathematical optimization, the local search (LS) is an iterated algorithm, where, starting from an initial solution and in each iteration, "neighbor" solutions to the current solution are searched to find better solutions in

terms of the cost. In case of a success in finding such a solution, the current solution will be updated, and the other iteration will begin to search in the neighbor of the new solution; otherwise, the local search stops with expressing the best found solution. In such an algorithm, the concept of neighborhood is derived from the definition of the neighborhood structure on each solution of the problem. The neighborhood of a given solution is usually a set of solutions that are obtained by applying one or more moves on the solution. The concept of a move will be carefully defined in the following discussions.

A local search algorithm often stops with the introduction of a local optimal solution. A simple idea to escape the local optima is to cleverly iterate the local search process, which will often help dispersing the search process in the solution space by starting from different initial solutions. Such an algorithm is often called an Iterated Local Search (ILS). In order to describe it precisely, consider the solution obtained from the local search as the current solution. In each round of the iterated local search algorithm, the current solution is updated using methods called the perturbation mechanisms. The updated solution is then improved by a local search to get a new solution. If the new solution is more appropriate than the current solution, the local search algorithm continues with the new solution; otherwise, the algorithm continues with the current solution. Figure 1 shows a stage of the ILS algorithm in which the local minimum $\hat{s}$ is perturbed and the solution s' is obtained, and then the local search is applied and a new local (possibly better) minimum ŝ' is found.

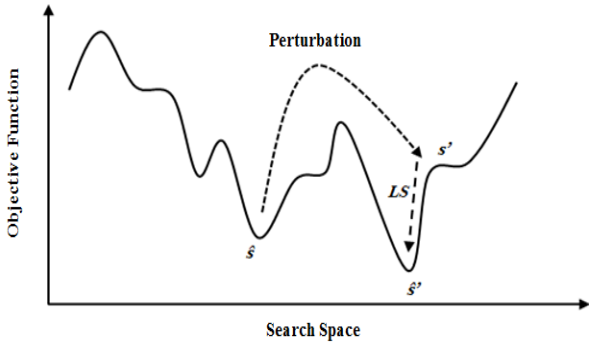

Figure 1. An iteration of the ILS algorithm

The overall structure of the iterated local search algorithm is as follows:

Step 1. Starting with an initial solution, run the local search method to find the current solution.
Step 2. If the stopping condition is met, stop with the introduction of the best solution found; otherwise, go to the next step.
Step 3. Choose the current solution by applying solution updates and perturbation mechanisms.
Step 4. Get a new solution by performing a local search on the current solution. If the new solution is better than the current solution, update the current solution and go to step 2.

4.1. Initial solution construction

The process of finding an initial solution is accomplished in three steps. In the first step, based upon the instances created, define a threshold between 0 and 1. Given the number of feasible locations for the controller installation in each instance, generate a random number between 0 and 1 for each location. If the random value generated for each location is greater than the threshold, then a proper controller is installed in the location. In the second step, the switches are assigned to the closest location containing a controller in accordance with constraint (6). Finally, in the third step, remove any controller to which no switch has been assigned.

4.2. Local search in solution space

The expression of a local search algorithm requires the definition of an appropriate neighborhood structure and the determination of the mechanism of a move from one solution to the neighboring solution in the solution space. In the following, the types of moves among the solutions are expressed.

Move Type 1. Consider a solution, *sol*, and a switch *s*. Assume that switch *s* is assigned to the controller located in *p*. Assigning switch *s* to a different location p' defines a move of type 1. In this allocation, if there is no controller at the location p', the new controller $c \in C$ is installed in location p' and switch *s* is assigned to it. If at the time of assigning the switch *s* to p' no more switches are assigned to *p*, the controller is removed from location *p*. The solutions generated by applying a move of type 1 in *sol* are called the neighbors type 1 of *sol*. The set of all such neighbors is denoted by F_{sol}^{1}.

Move Type 2. Consider a solution, *sol*, and two switches s and s'. Assume that switch *s* is assigned to the controller located *p* and switch s' to p'. The assignment of switch *s* to p' and s' to *p* defines a move of type 2. The solutions generated by

applying a move of type 2 in *sol* are called neighbors type 2 of *sol*. The set of all such neighbors is denoted by F_{sol}^2 .

Move Type 3. Consider a solution, *sol*, with the controller *c*. Suppose that *c* is located in *p*, and switch *s* is assigned to it. In this move, controller *c* will be removed from *p* and relocated in $p^{'}$. Then all switches connected to *p* will be removed and reconnected to $p^{'}$ accordingly. This assignment defines a move of type 3. The solutions generated by applying a move type 3 in *sol* are called the neighbors type 3 of *sol*. The set of all such neighbors is denoted by F_{sol}^3 .

Move Type 4. Consider a solution, *sol*, with the controller *c* and switch *s* from it. Assume that switch *s* is assigned to the controller *c* in the location *p*. The controller *c* is removed from the location *p* and switch *s* is assigned to the closest location containing a controller. This action defines a move of type 4. The solutions generated by applying a move type 4 in *sol* are called neighbors type 4 of *sol*. The set of all such neighbors is denoted by F_{sol}^4 .

Any move of the above-mentioned types in the current solution is considered as a *desired* move if it reduces the value of the objective function. In this case, a neighbor obtained by the move is called a *desired neighbor*. A best possible desired move is accordingly called an *optimal* move. Given different methods to move in the structure of a solution, the steps of a local search are as follow:

Step 1. Find an initial solution and consider it as the current solution, *sol*.

Step 2. If an optimal move is found in $F_{sol}^1 \cup F_{sol}^2 \cup F_{sol}^3 \cup F_{sol}^4$, update the current solution with an optimal neighbor and go to the beginning of step 2; otherwise, stop the search by returning to the current solution.

The above local search process is briefly called, the *search loop*. The runtime behavior of the local search algorithm depends on its neighborhood structure. For this purpose, the number of moves in this algorithm is limited to a certain value, say *L*.

4.3. Perturbation mechanisms of a solution

Many iterations of the search loop that are predicted in the ILS algorithm have an undesirable effect on its execution time. An essential idea in such a situation is the concentrated implementation of the search loop. If there is a possibility to execute a search loop on a specific part of the solution space, often a smaller number of moves will be required to find the suitable solutions in that part. One of the methods available to achieve these goals is to use the perturbation mechanisms of the problem solution. Perturbation mechanisms are divided into two categories: global and non-global. In a global perturbation, the solution structure changes to a great extent. In contrast, a non-global perturbation uses the kind of locking mechanism. This method makes it possible to keep fixed parts of the current solution during execution of the next search loop. The global and non-global perturbation mechanisms that apply to a current solution are as follow:

4.3.1. Non-global perturbation mechanisms

Given a solution to the problem:

Type 1. Randomly select a location, say *p*, to which only one switch is connected. Remove the controller from *p* and re-assign the switch to the closest controller. In the next run of the search loop, location *p* is locked, i.e. no controller will be installed there.

Type 2. Randomly select a location, say *p*, with no controller installed. Install a controller at *p*. Then from the set of all available switches, re-assign those that are closer to *p* than their own controller. In the next run of the loop, location *p* is locked, i.e. the controller installed will not be removed.

Type 3. Randomly select a location containing a controller and a location without a controller. Name them *p* and $p^{'}$ respectively. Then remove the controller from *p* and install a controller at $p^{'}$. Re-assign all switches connected to *p* to $p^{'}$. In the next run of the loop, locations *p* and $p^{'}$ are locked.

Type 4. Assume that $\tau = \sum_{s \in S} \sigma^s$, $\phi = \sum_{\substack{p \in P \\ c \in C}} \mu^c x_{cp}$, $P^{'} = \{p \in P : x_{cp} = 1\}$ and $P^{''} = \frac{P}{P^{'}}$. Choose a location with a controller and two locations without a controller. Name them *k*, l_1 and l_2, respectively. If

$$\min\{-f_h + f_r + f_w : \phi - \mu_h^c + \mu_r^c + \mu_w^c - \tau \geq 0,$$
$$h \in P^{'}, r \in P^{''}, w \in P^{''}\} < 0$$

then remove the controller from location and install the controllers in locations l_1 and l_2. The

combination of locations k, l_1 and l_2 is determined as follows:

$$(k, l_1, l_2) = \arg\min\{-f_h + f_r + f_w : \phi - \mu_h^C + \mu_r^C + \mu_w^C - \tau \geq 0,$$
$$h \in P^{'}, r \in P^{''}, w \in P^{''}\}.$$

Re-assign the switch connected to location K to the nearest location between l_1 and l_2. In the next run of the search loop, the locations k, l_1 and l_2 are locked, and the installation of the controller in location k and the removal of the controller in locations l_1 and l_2 will not be possible.

Type 5. Select two locations containing a controller and one location without a controller. Name them, l_1, l_2 and k, respectively.

If

$$\min\{f_h - f_r - f_w : \phi + \mu_h^C - \mu_r^C - \mu_w^C - \tau \geq 0,$$
$$h \in P^{'}, r \in P^{''}, w \in P^{''}\} < 0$$

then install the controller in location k and remove from locations l_1 and l_2. The combination of locations k, l_1 and l_2 is determined as follows:

$$(k, l_1, l_2) = \arg\min\{f_h - f_r - f_w : \phi + \mu_h^C - \mu_r^C - \mu_w^C - \tau \geq 0,$$
$$h \in P^{'}, r \in P^{''}, w \in P^{''}\}.$$

Re-assign the switches connected to the locations l_1 and l_2 to the location k. In the next run of the search loop, the locations k, l_1 and l_2 are locked, and there is no possibility to remove the controller from the location k and install the controller in locations l_1 and l_2.

4.3.2 Global perturbation mechanisms

Given a solution to the problem:

Type 1. First, remove the connection between all switches and controllers. Then randomly re-assign all the switches to locations containing the controller.

Type 2. The set $P^{'}$ contains all locations containing a controller; $P^{'} = \{p \in P : x_{cp} = 1\}$. Randomly put $\left\lceil \frac{|P|}{2} \right\rceil$ of members of $P^{'}$ in a set called O and the others in $O^{'}$. As a result, $|O| = \left\lfloor \frac{|P|}{O} \right\rfloor$. Remove the controllers from $O^{'}$ and re-assign the switch in $k = \{s \in S : v_{sp}^{l} = 1 \cap p \in O^{'} \cap l \in L\}$ to the controllers of set O; in other words, the switches $s \in k$ is re-assigned to locations O that has number of enough port; otherwise, s is re-assigned to the location $t = \arg\min_{p \in P \setminus O} f_p$ and x_t =1. In the following, the update mechanisms of the current solution are expressed based on the local search result.

Perturbation mechanisms are applied to a solution to achieve a search diversification. Usually, perturbation is done by randomly applying one of the local perturbation operators to a solution. However, these operators might not always achieve the right diversification effect. Hence, after a specified number of consecutive iterations without improvement to the best known solution, some stronger perturbation operators (i.e. global operators) are used to perturb a solution.

4-4 Solution update mechanisms

The iterated local search algorithm uses mechanisms to update the current solution immediately before the implementation of perturbation mechanisms. A simple mechanism for updating the current solution is to select the best solution from the previous search loop as a new current solution. Another is the replacement of the most favorable global solution found instead of the current solution. Another mechanism is to use a perturbation of one of the previous mechanisms. That means that, before replacing a solution in the current solution, limited movements will be applied to the perturbation. Another mechanism is the use of a random combination of the previous mechanisms in updating the current solution.

4.5 Termination condition

The iterated local search algorithm ends after repeating steps 3 and 4 a pre-specified number of time.

5- Simulation results

In this section, the experiments and computations done to evaluate the problem solving methods of controller placement problem are reported. Here, the runtime and quality of the solution for each problem solving method, CPLEX [18,19,20], SCIP [21,22,23], and Iterated Local Search algorithm, on a set of standard instances of problem, are calculated and compared with each other. CPLEX and SCIP are both well-known Branch and Bound solvers of mixed integer programming problems. CPLEX is a commercial software package, while SCIP is a non-commercial one. Indeed, extensive studies in mathematical programming literature would suggest CPLEX as the most powerful commercial, and SCIP as the most powerful non-commercial one [21]. In the following, we describe how to create problem instances:

For each instance, the network topology is randomly extracted from a 20 × 20 grid. That means that each node of the grid with probability p_r is a node of network graph. The value of p_r for instances of a medium size is set to 0.25 and for

instances of a large size is set to 0.5. After selecting the nodes, the complete graph will be induced to the instance. On this graph, the weight of each edge is defined by the Euclidean distance between the end nodes of the edge. In the next step, the nodes containing the switch on the graph are specified. For an instance with *i* switches, *i* nodes of the graph are randomly selected, and on each one of them a switch is installed.

The parameter *i*, number of switches installed on an instance, is taken from {55+5k | k=0, 1…15}. Instances made with i<95 are of medium size and instances with i≥95 are of large size. In order to verify more precisely, for each *i*, five instances with the same topology and different locations of the switches will be generated, and the results of the experiments will be reported on them. These instances are named i_1 … i_5.

All computations of this section are performed on an Intel core i5 under Windows operating system with 4 GB of main memory. The proposed ILS algorithm is implemented by MATLAB. For CPLEX and SCIP, the time limit is considered equal to 3600 seconds. The defined parameters for problem solving in CPLEX, SCIP, and the proposed ILS algorithm are given in table 1.

Table 1. Problem solving parameters.

Parameter name	Value
Cost per controller(k^c)	2500 $
Number of ports per controller(α^c)	32
Number of packages of process able by the controller(μ^c)	4000
Number of existing controllers(φ^c)	15
Link cost(Φ^l)	8.25 $
Link bandwidth(ω^l)	100 Mbps
Packet size(β)	150 Byte
Maximum network latency(γ)	250 ms
Average packet processing time by controller(δ)	0.001 ms

In these experiments, we set L = 10. The results of these experiments are summarized in tables 2 to 5. In these tables, for each instance, the following items are reported:

Cost: The best cost on these instances (cost of the best solution found)

Time: The time spent on these instances.

%imp: Percentage improvement of the ILS algorithm over CPLEX or SCIP in terms of the best cost on these instances, calculated as:

$$\frac{\text{Cost}_{CPLEX\ or\ SCIP}(\rho) - \text{Cost}_{ILS}(\rho)}{\text{Cost}_{ILS}(\rho)} * 100 \qquad (16)$$

In calculating this quantity, the $\text{Cost}_{ILS}(.)$ and $\text{Cost}_{CPLEX\ or\ SCIP}(.)$ functions give the best cost obtained from the implementation of the proposed ILS algorithm and the best solution returned by CPLEX or SCIP.

Table 2. Results obtained from ILS algorithm and CPLEX on instances of medium size.

Instance	CPLEX		ILS		CPLEX vs. ILS
Switch	Time	Cost ($)	Time	Cost ($)	Imp (%)
50_1	>3600	2060402	83/10	2156934	-4/48
50_2	>3600	2083714	85/01	2121060	-1/76
50_3	>3600	1977563	82/23	2032568	-2/71
50_4	>3600	2057832	76/15	2057015	0/04
50_5	>3600	2158843	74/10	2121134	1/78
55_1	>3600	2141661	72/70	2152656	-0/51
55_2	>3600	2128101	82/77	2124522	0/17
55_3	>3600	2160467	70/15	2138024	1/05
55_4	>3600	2232948	63/67	2184970	2/20
55_5	>3600	2138262	71/12	2081748	2/71
60_1	>3600	2368942	80/94	2279755	3/91
60_2	>3600	2335858	88/19	2295361	1/76
60_3	>3600	2311958	78/14	2282079	1/31
60_4	>3600	2311958	78/67	2282079	1/31
60_5	>3600	2324635	78/70	2339523	-0/64
65_1	>3600	2474250	88/98	2482917	-0/35
65_2	>3600	2535529	87/36	2556031	-0/80
65_3	>3600	2469251	90/90	2513834	-1/77
65_4	>3600	2546134	114/60	2568095	-0/86
65_5	>3600	2392448	104/64	2494901	-4/11
70_1	>3600	2571290	111/85	2623449	-1/99
70_2	>3600	2676821	144/85	2616256	2/31
70-3	>3600	2568866	117/44	2622997	-2/06
70_4	>3600	2721114	103/64	2633592	3/32
70_5	>3600	2640908	99/11	2603592	1/43
75_1	>3600	2805809	117/78	2784083	0/78
75_2	>3600	2722786	108/70	2761407	-1/40
75_3	>3600	2805857	105/04	2850915	-1/58
75_4	>3600	2843451	93/19	2840859	0/09
75_5	>3600	2880774	117/31	2786828	3/37
80_1	>3600	2969892	138/76	2912391	1/97
80_2	>3600	3035377	112/13	3017658	0/59
80_3	>3600	2966433	115/04	2932684	1/15
80_4	>3600	2957112	111/41	2940124	0/58
80_5	>3600	2944530	111/54	2932955	0/39
85_1	>3600	3041014	121/42	3047510	-0/21
85_2	>3600	3132328	121/80	3119139	0/42
85_3	>3600	3040845	134/15	3024267	0/55
85_4	>3600	3129574	119/66	3061835	2/21
85_5	>3600	3129574	120/08	3061835	2/21
90_1	>3600	3299413	131/25	3171293	4/04
90_2	>3600	3398409	206/76	3251215	4/53
90_3	>3600	3343441	129/96	3199939	3/67
90_4	>3600	3229046	130/41	3140596	1/03
90_5	>3600	3249320	146/37	3145293	1/03

Table 3. Results obtained from ILS algorithm and CPLEX on instances of large size.

Instance	CPLEX		ILS		CPLEX vs. ILS
Switch	Time	Cost ($)	Time	Cost ($)	Imp (%)
95_1	3600>	3627266	139/47	3359813	7/96
95_2	3600>	3621218	138/69	3377347	7/22
95_3	3600>	3541875	139/27	3300338	7/32
95_4	3600>	3596935	141/11	3350564	7/35
95_5	3600>	3600972	138/66	3328449	8/19
100_1	3600>	3705810	237/83	3465648	6/93
100_2	3600>	3685675	291/96	3513347	4/90
100_3	3600>	3770711	292/83	3851943	-2/11
100_4	3600>	3712408	285/43	3514685	5/63
100_5	3600>	3462436	224/16	3356697	3/15
105_1	3600>	4051630	2081/65	3734200	8/50
105_2	3600>	4048558	1641/11	3752400	7/89
105_3	3600>	4048087	1523/86	4222601	-4/13
105_4	3600>	4201017	2270/32	3785708	10/97
105_5	3600>	4295851	1799/67	4057574	5/87
110_1	3600>	4245739	1978/19	3901900	8/81
110_2	3600>	4344531	1552/55	4085242	6/35
110_3	3600>	4350938	1729/84	3882728	12/06
110_4	3600>	4271414	1608/30	4109212	3/95
110_5	3600>	4285425	1779/55	3832800	11/81
115_1	3600>	4457508	2369/27	4190600	6/37
115_2	3600>	4462837	1687/27	4055076	10/06
115_3	3600>	4361320	1645/61	4119705	5/86
115_4	3600>	4467130	1838/65	3974471	12/40
115_5	3600>	4467130	1422/99	3974471	12/40
120_1	3600>	4935663	1601/79	4205000	17/38
120_2	3600>	4955742	2006/49	4263500	16/24
120_3	3600>	4876055	1719/99	4383961	11/22
120_4	3600>	4811952	1521/64	4373944	10/01
120_5	3600>	4746181	1280/93	4306507	10/21
125_1	3600>	5113260	1755/23	5605200	-8/78
125_2	3600>	5098265	2401/76	4824409	5/68
125_3	3600>	5299716	2587/37	4873091	8/75
125_4	3600>	5144576	1378/53	4682895	9/86
125_5	3600>	4972032	1173/50	4598856	8/11
130_1	3600>	5708434	3303/53	5259600	8/53
130_2	3600>	5532203	2406/87	4983800	11/00
130_3	3600>	5514490	1750/34	4972800	10/89
130_4	3600>	5503903	1748/84	4895352	12/43
130_5	3600>	5490216	1253/54	4891089	12/25

Table 4. Results obtained from ILS algorithm and SCIP on instances of medium size.

Instance	SCIP		ILS		SCIP vs. ILS
Switch	Time	Cost ($)	Time	Cost ($)	Imp (%)
50_1	>3600	2101445	83/10	2156934	-2/57
50_2	>3600	2086498	85/01	2121060	-1/63
50_3	>3600	1997456	82/23	2032568	-1/73
50_4	>3600	2003364	76/15	2057015	-2/61
50_5	>3600	2037644	74/10	2121134	-3/94
55_1	>3600	2137956	72/70	2152656	-0/68
55_2	>3600	2123838	82/77	2124522	-0/03
55_3	>3600	2166462	70/15	2138024	1/33
55_4	>3600	2205382	63/67	2184970	0/93
55_5	>3600	2105740	71/12	2081748	1/15
60_1	>3600	2274063	80/94	2279755	-0/25
60_2	>3600	2362716	88/19	2295361	2/93
60_3	>3600	2283537	78/14	2282079	0/06
60_4	>3600	2283537	78/67	2282079	0/06
60_5	>3600	2290199	78/70	2339523	-2/11
65_1	>3600	2416234	88/98	2482917	-2/69
65_2	>3600	2470685	87/36	2556031	-3/34
65_3	>3600	2419417	90/90	2513834	-3/76
65_4	>3600	2565170	114/60	2568095	-0/11
65_5	>3600	2405980	104/64	2494901	-3/18
70_1	>3600	2555790	111/85	2623449	-2/58
70_2	>3600	2597144	144/85	2616256	-0/73
70-3	>3600	2566105	117/44	2622997	-2/17
70_4	>3600	2609453	103/64	2633592	-0/92
70_5	>3600	2610128	99/11	2603592	0/25
75_1	>3600	2753493	117/78	2784083	-1/10
75_2	>3600	2753297	108/70	2761407	-0/29
75-3	>3600	2795299	105/04	2850915	-1/95
75_4	>3600	2778696	93/19	2840859	-2/19
75_5	>3600	2771459	117/31	2786828	-0/55
80_1	>3600	2888153	138/76	2912391	-0/83
80_2	>3600	2989984	112/13	3017658	-0/92
80_3	>3600	2956982	115/04	2932684	0/83
80_4	>3600	2956103	111/41	2940124	0/54
80_5	>3600	2964306	111/54	2932955	1/07
85_1	>3600	3039302	121/42	3047510	-0/27
85_2	>3600	3108805	121/80	3119139	-0/33
85_3	>3600	3065483	134/15	3024267	1/36
85_4	>3600	3192355	119/66	3061835	4/26
85_5	>3600	3091573	120/08	3061835	0/97
90_1	>3600	3194227	131/25	3171293	0/72
90_2	>3600	3273411	206/76	3251215	0/68
90_3	>3600	3317222	129/96	3199939	3/67
90_4	>3600	3172976	130/41	3140596	1/03
90_5	>3600	3177639	146/37	3145293	1/03

Table 5. Results obtained from ILS algorithm and SCIP on instances of large size.

In tables 2 to 5, the first column gives the names of the instances tested. We recall that in this naming, the left-hand digit represents the number of switches used in the instance and the right-hand digits represent the instance number. The second column shows the running time of CPLEX and SCIP on the instances. The third column represents the cost of the best solution found by CPLEX and SCIP within a time limit of 3600 seconds. The fourth and fifth columns report the runtime and the cost of the best solution obtained by the ILS algorithm. Finally, the sixth column represents the percentage of improvement of the ILS algorithm compared to CPLEX and SCIP in terms of the cost of the best solution found.

The results obtained from tables 2 and 4 refer to the superiority of the proposed ILS algorithm at the runtime for all instances with a medium size and the best cost for some of these instances. Tables 3 and 5 also refer to the superiority of the proposed ILS algorithm, both at the runtime and at the best cost, in more instances with a large size.

Figures 2 and 3 show the average percentage improvement of the ILS algorithm compared to CPLEX and SCIP for different instances where the horizontal axis gives the switch size in each instance, and the vertical axis gives the average percentage improvement optimal. According to this chart, the superiority of the ILS algorithm gradually increases in finding solutions at a lower cost. Back to tables 2 to 5, ILS also achieves a much better running time behavior over those instances. This causes the proposed algorithm to be considered as an important tool for solving a large-scale problem.

Instance	SCIP		ILS		SCIP vs. ILS
Switch	Time	Cost ($)	Time	Cost ($)	Imp (%)
95_1	>3600	3499238	139/47	3359813	4/15
95_2	>3600	3513357	138/69	3377347	4/03
95_3	>3600	3439830	139/27	3300338	4/23
95_4	>3600	3508290	141/11	3350564	4/71
95_5	>3600	3493537	138/66	3328449	4/96
100_1	>3600	3648637	237/83	3465648	5/28
100_2	>3600	3681437	291/96	3513347	4/78
100_3	>3600	3783988	292/83	3851943	5/64
100_4	>3600	3623630	285/43	3514685	3/10
100_5	>3600	3496197	224/16	3356697	4/16
105_1	>3600	4247645	2081/65	3734200	13/75
105_2	>3600	4240761	1641/11	3752400	13/01
105_3	>3600	4404533	1523/86	4222601	4/31
105_4	>3600	4021226	2270/32	3785708	6/22
105_5	>3600	4377527	1799/67	4057574	7/89
110_1	>3600	4268660	1978/19	3901900	9/40
110_2	>3600	4555893	1552/55	4085242	11/52
110_3	>3600	4607072	1729/84	3882728	18/66
110_4	>3600	4460211	1608/30	4109212	8/54
110_5	>3600	4332976	1779/55	3832800	13/05
115_1	>3600	4769452	2369/27	4190600	13/81
115_2	>3600	4636152	1687/27	4055076	14/33
115_3	>3600	4681238	1645/61	4119705	13/63
115_4	>3600	4774883	1838/65	3974471	20/14
115_5	>3600	4774883	1422/99	3974471	20/14
120_1	>3600	5215515	1601/79	4205000	24/03
120_2	>3600	4854909	2006/49	4263500	13/87
120_3	>3600	5174666	1719/99	4383961	18/04
120_4	>3600	5074645	1521/64	4373944	16/02
120_5	>3600	5011034	1280/93	4306507	16/36
125_1	>3600	5606027	1755/23	5605200	0/01
125_2	>3600	5453273	2401/76	4824409	13/04
125_3	>3600	5755598	2587/37	4873091	18/11
125_4	>3600	5522270	1378/53	4682895	17/92
125_5	>3600	5244840	1173/50	4598856	14/05
130_1	>3600	5774233	3303/53	5259600	9/78
130_2	>3600	5995312	2406/87	4983800	20/30
130_3	>3600	5602644	1750/34	4972800	12/67
130_4	>3600	5816147	1748/84	4895352	18/81
130_5	>3600	5614192	1253/54	4891089	14/78

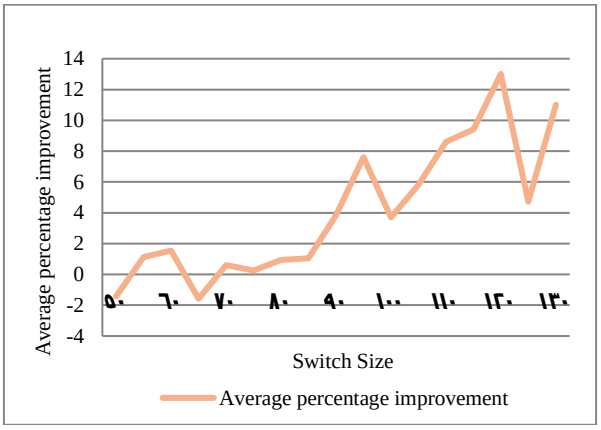

Figure 2. Average percentage of improvement of the ILS algorithm compared to CPLEX.

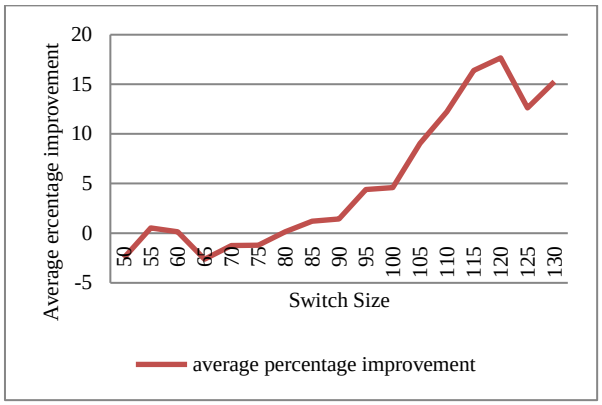

Figure 3. Average percentage of improvement of the ILS algorithm compared to SCIP.

Figures 4 and 5 show the average runtime of the ILS algorithm compared to CPLEX and SCIP. The horizontal axis in these figures shows the size of instances based on the number of installed switches and the vertical axis shows the average runtime on the instances. The results obtained clearly show that the proposed algorithm is able to find better solutions in a shorter time. This feature will express the proposed method as an effective method for solving the controller placement problem in real-world applications.

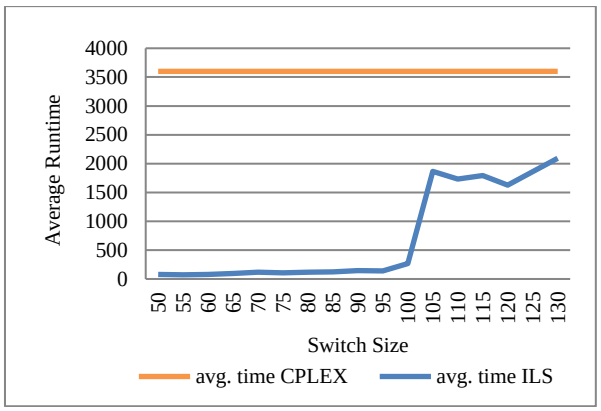

Figure 4. Average runtime of the ILS algorithm on the instances of the same size compared to CPLEX.

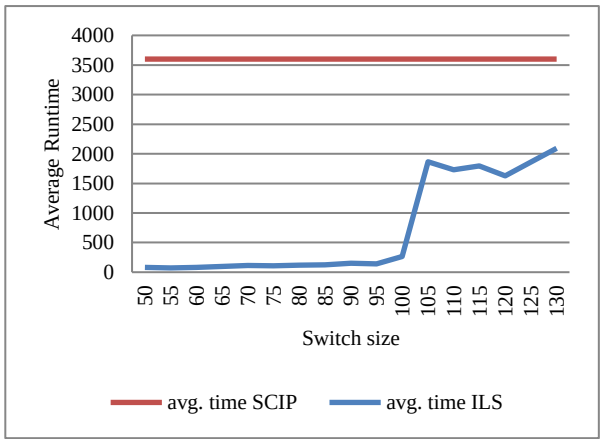

Figure 5. Average runtime of the ILS algorithm on the instances of the same size compared to SCIP.

6. Conclusion

In this work, the controller placement problem in the SDN networks has been studied. In order to solve this problem, an algorithm based on the iterated local search method was proposed. The proposed algorithm uses the concepts such as the neighborhood and perturbation mechanism to achieve an efficient topology at an acceptable running time. In order to evaluate the performance of the proposed algorithm, experiments were conducted on several instances of networks with medium to large sizes. The results obtained from the proposed ILS algorithm were compared with the results of CPLEX and SCIP performed on similar instances. The results obtained indicate the superiority of the proposed ILS algorithm at the runtime for all instances and at the best solution found in some instances of medium size. Also, in instances of a large size, the proposed ILS algorithm is superior both in the runtime and in the best cost found.

Reference

[1] Blial, O., Ben Mamoun, M., & Benaini, R. (2016). An overview on SDN architectures with multiple controllers, Journal of Computer Networks and Communications, vol. 2016.

[2] Selvi, H., Güner, S., Gür, G., & Alagöz, F. (2015). The controller placement problem in software defined mobile networks (SDMN), Software Defined Mobile Networks (SDMN): Beyond LTE Network Architecture, pp. 129-147.

[3] Nunes, B. A. A., Mendonca, M., Nguyen, X. N., Obraczka, K., & Turletti, T. (2014). A survey of software-defined networking: Past, present, and future of programmable networks, IEEE Communications Surveys & Tutorials, vol. 16, no. 3, pp. 1617-1634.

[4] Xia, W., Wen, Y., Foh, C. H., Niyato, D., & Xie, H. (2014). A survey on software-defined networking. IEEE Communications Surveys & Tutorials, vol. 17, no. 1, pp. 27-51.

[5] Open networking foundation. https://www.opennetworking.org/about.

[6] Jarraya, Y., Madi, T., & Debbabi, M. (2014). A survey and a layered taxonomy of software-defined networking, IEEE communications surveys & tutorials, vol. 16, no. 4, pp. 1955-1980.

[7] Sezer, S., Scott-Hayward, S., Chouhan, P. K., Fraser, B., Lake, D., Finnegan, J. & Rao, N. (2013). Are we ready for SDN? Implementation challenges for software-defined networks, IEEE Communications Magazine, vol. 51, no. 7, pp. 36-43.

[8] Heller, B., Sherwood, R., & McKeown, N. (2012). The controller placement problem, In Proceedings of the first workshop on Hot topics in software defined networks, ACM, pp. 7-12.

[9] Yao, G., Bi, J., Li, Y., & Guo, L. (2014). On the capacitated controller placement problem in software defined networks, IEEE Communications Letters, vol. 18, no. 8, pp. 1339-1342.

[10] Lourenço, H. R., Martin, O. C., & Stützle, T. (2019). Iterated local search: Framework and applications, In Handbook of metaheuristics, Springer, pp. 129-168.

[11] Xiao, P., Qu, W., Qi, H., Li, Z., & Xu, Y. (2014). The SDN controller placement problem for WAN, In 2014 IEEE/CIC International Conference on Communications in China (ICCC), IEEE, pp. 220-224.

[12] Hu, Y. N., Wang, W. D., Gong, X. Y., Que, X. R., & Cheng, S. D. (2012). On the placement of controllers in software-defined networks, The Journal of China Universities of Posts and Telecommunications, vol. 19, pp. 92-171.

[13] Zhang, Y., Beheshti, N., & Tatipamula, M. (2011). On resilience of split-architecture networks, In 2011 IEEE Global Telecommunications Conference-GLOBECOM 2011, IEEE, pp. 1-6.

[14] Obadia, M., Bouet, M., Rougier, J. L., & Iannone, L. (2015). A greedy approach for minimizing SDN control overhead, In Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft), IEEE, pp. 1-5.

[15] Yao, G., Bi, J., Li, Y., & Guo, L. (2014). On the capacitated controller placement problem in software defined networks, IEEE Communications Letters, vol. 18, no. 8, pp. 1339-1342.

[16] Zhang, T., Bianco, A., & Giaccone, P. (2016). The role of inter-controller traffic in SDN controllers placement, In 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE, pp. 87-92.

[17] Sallahi, A., & St-Hilaire, M. (2014). Optimal model for the controller placement problem in software defined networks, IEEE communications letters, vol. 19, no. 1, pp. 30-33.

[18] IBM Inc. IBM ILOG CPLEX optimization studio getting started with CPLEX (version 12 release 6). IBM Corporation; (2014). Available: https://www.ibm.com/analytics/cplex-optimizer

[19] Dowlatshahi, M. B., & Derhami, V. (2017). Winner Determination in Combinatorial Auctions using Hybrid Ant Colony Optimization and Multi-Neighborhood Local Search, Journal of AI and Data Mining, vol. 5, no. 2, pp. 169-181.

[20] Ashrafi, M., Correia, N., & Faroq, A. T. (2018). A Scalable and Reliable Model for the Placement of Controllers in SDN Networks, In International Conference on Broadband Communications, Networks and Systems, Springer, pp. 72-82.

[21] Solving constraint integer programs. http://scip.zib.de/

[22] Mueller, J., Wierz, A., & Magedanz, T. (2013). Scalable On-Demand Network Management Module for Software Defined Telecommunication Networks, In SDN4FNS, pp. 1-6.

[23] Herbaut, N., Negru, D., Magoni, D., & Frangoudis, P. A. (2016). Deploying a content delivery service function chain on an SDN-NFV operator infrastructure, In 2016 International Conference on Telecommunications and Multimedia (TEMU), IEEE, pp. 1-7.

[24] Stützle, T., & Ruiz, R. (2017). Iterated local search. Handbook of Heuristics, pp. 1-27.