Variable Depth Search and Iterated Local Search for the Node Placement Problem in Multihop WDM Lightwave Networks

Kengo Katayama, Member, IEEE, Hiroshi Yamashita, Hiroyuki Narihisa

Abstract—We address a problem of finding an optimal node placement that minimizes the amount of traffics by reducing the weighted hop distances in multihop lightwave networks. The problem is called Node Placement Problem (NPP). NPP is known to be NP-hard and one of the most important problems in wavelength division multiplexing (WDM) based networks.

In this paper we propose a new local search algorithm for the NPP based on variable depth search, and show its extension to an iterated local search algorithm. To evaluate the performance of the proposed methods, we provided the benchmark instances with known optimal solutions, and performed extensive experiments on the instances. The computational results showed that our iterated local search outperformed multistart local search methods and the best available metaheuristic for the problem.

I. INTRODUCTION

We consider a combinatorial optimization problem, called Node Placement Problem (NPP for short), motivated by applications in multihop lightwave networks. The NPP is known to be NP-hard [1], [2].

The network has a regular topology that can be represented by a grid graph on torus $G_{m,n} = (V,E)$, where $V$ is the set of node slots that form $m$ rows and $n$ columns ($n = m \times m$) and $E$ is the set of bidirectional edges. Let $(x,y)$ denotes a slot address (coordinates) in the $x$-th row and the $y$-th column ($x,y = 0,1,\ldots,n-1$) of the graph. Each node slot has four bidirectional edges in vertical and horizontal directions as so to form a torus.

Each of $n$ nodes $(0,1,\ldots,n-1)$ can be assigned to each of the $n$ slots on the graph without duplications, and the amount of traffics among network nodes can be given by an $n \times n$ traffic matrix $T$, where each element $t_{i,j}$ denotes the traffic flow from node $i$ to node $j$ ($i,j = 0,1,\ldots,n-1$, $i \neq j$) that has a real or integer value. We simplify the amounts into two types of traffics (heavy and light, denoted by $t_H$ and $t_L$, respectively) as follows:

$$t_{i,j} = \begin{cases} 1 & \text{if nodes } i \text{ and } j \text{ are highly communicated} \\ 0 & \text{otherwise.} \end{cases}$$

The diagonal is off, $t_{i,i} = 0$ for all $i$.

Assuming that $t_{i,j} = 1$, the node $i$ can communicate with the node $j$ directly if the nodes are assigned in adjacent slots of the graph. If not adjacent, they must communicate through several node slots, i.e., the number of hops increase. A function $h(i,j)$ is provided that returns the hop distances in the shortest path between two nodes $i$ and $j$ on the graph.

The objective of NPP is to find a node placement $\sigma$ of $n$ nodes that minimizes the weighted hop distances:

$$f(\sigma) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} t_{i,j} \times h(\sigma_i, \sigma_j),$$

where $\sigma_i (i = 0,1,\ldots,n-1)$ represents $i$-th node in the node placement that corresponds to the node number assigned at $([i/m], [i \mod m])$ slot coordinates in the graph of the network. Figure 1 shows an example of node placements for graph $G_{4,4}$, and the corresponding node placement is $\sigma = \{1,2,6,14,8,0,3,7,5,9,12,4,15,1,10,13\}$. If the traffic matrix gives that for example, nodes 8 and 10 (that are assigned at slots (1,0) and (3,2), respectively) are highly communicated (i.e., $t_{8,10} = 1$), the number of hops in the shortest path is 4 in this node placement.

The NPP defined above is equivalent to the problem of finding an optimal wavelength assignment for traffics given in the Bidirectional Manhattan Street Network (BMSN) [2], [3]. BMSN is one of the multihop lightwave networks based on Wavelength Division Multiplexing (WDM) that is a promising technology for high-speed communication, in which the vast bandwidth of a optical fiber is divided into multiple channels. Details of WDM and network topologies can be found in [4], [5], [6]. Recently, similar optimization problems have been investigated for other networks, e.g., ShuffleNets [7], [8].

Several (meta)heuristic algorithms have been proposed for solving the NPP: Kato and Oie investigated a greedy method, local search, tabu search, genetic algorithm, simulated annealing, threshold acceptance, and multistart local search [2]. They reported that the tabu search with the greedy method is the best one in their comparisons. Kitani, et al. proposed a new simulated annealing approach called HIWAS (Hierarchical Wavelength Assignment algorithm) in

---

K. Katayama, H. Yamashita, H. Narihisa are with the Department of Information and Computer Engineering, Okayama University of Science, 1 - 1 Ridaicho, Okayama, Japan (email: katayama, yamashita, narihisa)@ice.ous.ac.jp.

Fig. 1. A grid graph with 16 nodes (4 x 4)
which at first creates an initial solution hierarchically and the initial solution is improved by simulated annealing [3]. They showed that HIWAS obtains better solutions with shorter running times than the tabu search [2].

Most of the (meta)heuristic algorithms in the above studies [3], [2] for the NPP are based on neighborhood search using \textit{swap neighborhood}. This neighborhood can be defined, given a node placement (solution), as the set of neighbors that can be obtained by swapping a pair of nodes. Therefore, the basic principle of the neighborhood search algorithms is to swap two nodes in the solution at each iteration in order to find better node placements, and this principle based on the single-swap (1-swap) operation is repeated with many iterations until predefined stopping conditions are satisfied.

In this paper, we propose a new local search for the NPP, called $k$-swap local search (KLS). Our principle of KLS is to swap $2: k$ nodes ($k \geq 1$) simultaneously at each iteration by applying a sequence of the single-swap moves, borrowed from variable depth search [9], [10]. We also show an iterated local search metaheuristic embedded with KLS, called iterated $k$-swap local search (IKLS). To evaluate the performances of KLS and IKLS, we provided the benchmark instances and performed computational experiments on the instances. The results showed that IKLS outperforms multi-start local search methods and the simulated annealing [3] that is the best available metaheuristic for the problem.

II. LOCAL SEARCH AND VARIABLE DEPTH SEARCH

Local search [11] is a generally applicable approach that can be used to find approximate solutions to hard optimization problems. Many powerful metaheuristics [12], [13] such as iterated local search [14], memetic algorithm [15], etc. are based on it. In this section, we describe the basic process of local search and review the variable depth search.

A. Basic Process of Local Search

Given a feasible solution $x$ for a combinatorial minimization problem and the cost function $f$, the basic idea of local search is to start from a solution $x$ and to repeatedly replace $x$ with a better one $x'$, i.e. $f(x') < f(x)$, which is selected from the \textit{neighborhood} $\mathcal{N}(x)$ defined as the set of neighbors that can be reached by making slight changes to $x$. If no better neighbor solutions can be found in its neighborhood, local search immediately stops and returns as the best solution found during the search. The resulting solution can not be improved by such slight changes. It is called \textit{locally optimal} with respect to the defined neighborhood.

In local search the replacement of the current (incumbent) solution $x$ is generally divided to two move strategies so-called \textit{best improvement} and \textit{fist improvement}. The \textit{best improvement} selects the neighbor solution with the best cost in the entire candidate set of $\mathcal{N}(x)$, and the neighbor solution $x'$ becomes a new incumbent solution. The \textit{first improvement} scans neighbor solutions in $\mathcal{N}(x)$ according to a prespecified (random) order. If an improved neighbor $x'$ is found during the scan, the solution $x'$ is immediately accepted as a new incumbent solution.

Another factor affecting the quality of local optimal solutions (and running times consumed by local search) is the size of neighborhood. Although the larger-sized neighborhoods might yield better local optima, the effort expended to search the neighborhood is too computationally expensive.

B. Variable Depth Search Methods

To keep the computation time within a reasonable limit when considering a local search with the larger-sized neighborhoods, we can use \textit{variable depth search} (VDS), a well-known generalization of local search methods. The idea of VDS was first applied by Lin and Kernighan to the traveling salesman problem (TSP) [9] and graph partitioning problem (GPP) [10]. Their methods are often called \textit{Lin-Kernighan heuristic} for TSP and \textit{Kernighan-Lin heuristic} for GPP, or synthetically, \textit{(variable) $k$-opt local search}. The most important mechanism in VDS is to apply a sequence of moves to a given solution rather than only one move at each iteration. Because the length of the sequence may change from iteration to iteration in the algorithm, the size of neighborhood per iteration is variable. Thus, the basic concept is to change the size of the neighborhood adaptively so that the algorithm can effectively traverse larger search space within reasonable time.

Recently, for TSP and GPP, the VDS-based heuristics have been incorporated into several metaheuristic frameworks, such as iterated local search [16], [17], [18], [19] and evolutionary algorithm [20], [21], [22]. The general performance of metaheuristics embedded with VDS-based local search is remarkably effective for the hard problems TSP and GPP.

VDS is a general optimization method suitable for a wide range of combinatorial optimization problems. Successful applications of VDS so far include several problems. Murthy, et al. [23] showed a local search based on VDS for the quadratic assignment problem. Yagiura, et al. [24] suggested an algorithm for the generalized assignment problem. Tiourine, et al. [25] showed VDS algorithms for the radio link frequency assignment problem. For the unconstrained binary quadratic programming problem (UBQP), an effective local search based on VDS was proposed in [26] and then a variant method was suggested in [27]. Merz and Katayama [28] recently proposed a memetic algorithm with the variant VDS-based local search for UBQP and reported that the memetic algorithm is highly effective. More recently, a VDS-based algorithm, called $k$-opt local search (KLS), was proposed for the maximum clique problem (MCP) [29]. KLS was introduced to a framework of iterated local search [30], and the impressive results have been shown on DIMACS benchmark graphs of MCP.

Judging from these contributions, we can expect with metaheuristics embedded with VDS-based local search algorithms to offer promising approaches to other hard problems, such as the NPP considered in this paper.

III. LOCAL SEARCH ALGORITHMS FOR NPP

The performance of local search highly depends on neighborhood definitions and replacement ways as shown above.
In this section, we describe two local search methods called \textit{Best Improvement 1-swap Local Search} and \textit{First Improvement 1-swap Local Search} based on the swap neighborhood. We then show a new, effective one called \textit{(variable) k-swap Local Search} based on variable depth search for the NPP.

\subsection*{A. 1-swap Local Search Algorithms}

The 1-swap neighborhood $N_1$-swap can be defined, given a node placement $\sigma$, as the set of neighbors $\sigma'$ that can be obtained by swapping a pair of nodes $i$ and $j$, where $0 \leq i < j \leq n-1, i \neq j$. The size of $N_1$-swap is $n(n-1)/2$.

Local search based on $N_1$-swap is classified into \textit{Best Improvement 1-swap Local Search (1LS-BI)} and \textit{First Improvement 1-swap Local Search (1LS-FI)} that are characterized with each of two move strategies described above.

The pseudo-codes of the local search algorithms are shown in Figures 2 and 3, respectively. We assume that a feasible solution $\sigma$ (and traffic matrix $T$ of a given problem instance tacitly) is given beforehand for each local search. The function of $\text{SwapGain}(i,j,\sigma)$ in the figures returns a cost difference (gain value) $\delta(i,j)$ (per call) between the current solution $\sigma$ and the neighbor one $\sigma'$ that can be reached by swapping nodes $i$ and $j$ in $\sigma$. If a gain value of $\delta(i,j)$ is smaller than zero, it indicates an improvement. When the nodes $i$ and $j$ to be swapped are determined, the function $\text{SwapMove}(i,j,\sigma)$ is performed to create the neighbor solution $\sigma'$ from the current one $\sigma$.

The significant difference between 1LS-BI and 1LS-FI is only the process of the line 2 in Figures 2 and 3.

In each iteration of 1LS-BI, the entire neighbors are checked with their gains in the current solution, and the node pair with the best gain is selected. If the best neighbor gives an improvement, the local search is repeated after the replacement, otherwise, it is terminated.

On the other hand, in each iteration of 1LS-FI, a node pair is randomly selected with the gain calculation. If the current solution is improved by swapping the node pair, the neighbor solution is immediately replaced, and the local search is repeated until no better neighbors can be found. In our 1LS-FI, node $i$ is randomly selected without duplications of the range $0 \leq i \leq n - 1$, but node $j$ is given with the systematic order from 0 to $n-1$ because of reducing the number of calls of a random generator. It also contributes to the reduction in computation times. To further reduce the computation times without large loss of solution quality, we use "don't look bit" (DLB) [31]. DLB is known to be a useful mechanism for speeding up local search with the first improvement move strategy for various permutation problems such as the TSP [18], [13]. In our experiments, we also show the usefulness of DLB in 1LS-FI for the NPP.

All the algorithms shown in this paper, including the following k-swap local search can be implemented more efficiently by using problem instance characteristics. The calculation time of the solution cost using Eq. (2) takes $O(n^2)$ due to the traffic matrix $T$ of the size $n \times n$. If for each node $i$ in $\{0, 1, \ldots, n-1\}$ a heavy traffic information list of $j$'s for which $t_{i,j} \neq 0$ is maintained instead of the entire list of $n^2$, their running times can be reduced for instances with lower densities. Since the similar techniques can be used in the gain calculation process in the local search methods, the efficiency is increased considerably.

\subsection*{B. Variable k-swap Local Search Algorithm}

A single swap moving, i.e., 1-swap neighborhood, for a current node placement (solution) per iteration has been discussed in the above subsection. It is possible to consider larger neighborhoods such as multi-swap (i.e., $k$-swap) neighborhood that is the set of neighbors which can be obtained by swapping $2 \cdot k$ nodes ($1 \leq k \leq n/2$) simultaneously at each iteration for the NPP. In general, the larger the value of $k$, the more likely it is that the final solution will be optimal. This intuitively appears when $k$ is large ($1 < k \leq n/2$). However, it is impractical to obtain all neighbor solutions that can be reached by moves of a large, fixed $k$ swaps because the neighborhood size of a complete $k$-swap local search grows exponentially with $k$.

In order to remove the above issue of the exponential size of the complete $k$-swap neighborhood, we here propose a \textit{variable k-swap local search} (KLS) based on variable depth search (VDS) for the NPP. KLS determines dynamically at each iteration the value of $k$ (the number of swaps of nodes), since it is computationally too expensive to search the complete $k$-swap neighborhood. In KLS, the (variable) $k$-swap neighborhood $N_{k}$-swap of a given node placement $\sigma$ is defined as the set of chained neighbors $\sigma'$ that can be obtained by applying a sequence of the single-swap moves $\sigma$ to feasible search space. It indicates that KLS attempts to search a small fraction of the large neighborhoods in reasonable times. The length $l$ of the sequence is adaptively decided in the algorithm. All $l$ chained neighbor solutions obtained by the sequence are different because we assure that cycling among the neighbors in the sequence is avoided.

\begin{verbatim}
procedure Best Improvement 1-swap Local Search (σ)
begin
  repeat
    find a node pair (i, j)
    with \text{SwapGain}(i,j,σ) := \text{SwapGain}(i,j,σ),
    if \text{SwapGain}(i,j,σ) < 0 then \σ := \text{SwapMove}(i,j,σ);
    until no better neighbors can be found, i.e., \text{SwapGain}(i,j,σ) ≥ 0;
  return \σ;
end;

Fig. 2. Best Improvement 1-swap Local Search for NPP

procedure First Improvement 1-swap Local Search (σ)
begin
  repeat
    create a random pair (i, j) with 0 ≤ i < j ≤ n - 1, i ≠ j,
    and calculate \text{SwapGain}(i,j,σ);
    if \text{SwapGain}(i,j,σ) < 0 then \σ := \text{SwapMove}(i,j,σ);
    until no better neighbors can be found;
  return \σ;
end;

Fig. 3. First Improvement 1-swap Local Search for NPP
\end{verbatim}
This KLS with the hybrid scheme is faster at the sacrifice of solution quality than the standard VDS-based algorithm with only the best improvement move strategy. To further reduce the running time of KLS, the termination condition of the inner loop is modified so that the loop is terminated if the current gain $g$ is larger than the best gain value $g_{\text{LastImp}}$ recorded at the previous last iteration as shown at line 11 in Figure 4. The related processing can be found at lines 1 and 12. This modification is quite useful to considerably increase the efficiency of KLS without large loss of solution qualities. Note that no parameter setting by user is required for KLS.

IV. ITERATED LOCAL SEARCH FOR NPP

Iterated local search (ILS) [14] is a simple approach to combinatorial optimization problems, and is known to be a special case of memetic algorithms (MAs) [15]. Unlike MAs, ILS consists of only local search and mutation (called kick) operators. Although MA is one of the population-based metaheuristic methods in evolutionary computation that have many solutions for the search, ILS methods use a single solution in many cases. It indicates that ILS methods with fewer operators have fewer parameters set by user than the case of MAs (and other metaheuristic approaches) usually.

A. Basic Frameworks for Enhancement of Local Search

Local search can be trapped in local optima and be unable to reach the global optima. To reach the global optimum or if desired very good approximate solutions, local search (or VDS-based heuristic) can be enhanced in some sense. Multi-start Local Search (MLS) approach is one of the simplest enhancements, in which local search is repeatedly applied to newly generated solutions at random, and the best overall solution is kept and output as the result.

Iterated Local Search (ILS) [14] also provides a very simple framework to enhance local search for various hard optimization problems. ILS first generates a random solution, and then the solution (or mutated solution) is locally optimized by local search to obtain a locally optimal solution. The locally optimal solution (or previously found best one) is slightly perturbed by a mutation technique to obtain a mutated solution. These processes except for generating the initial solution in the first step are repeated until predefined termination conditions are satisfied. Thus, a general ILS framework can be simply composed of two processes in each iteration: a local search and a mutation (kick).

The role of kick process in ILS is to escape from local optimum found by local search process by moving to other points where are not so far from the local optima in the search space. This moving is made by perturbing the previously found local optimum slightly so that a different local optimum can be found by the forthcoming local search. An aspect of the kick process can be considered as a kind of neighborhoods, and we usually require a neighborhood structure that differs from one used in local search process. If we adopt the same (or quite similar) structure in both of local search and kick processes, the same local optimum would be reproduced by local search in a high probability. Therefore, the general

---

Procedure Variable k-swap Local Search ($\sigma$)

begin
1 $\sigma_{\text{best}} := \sigma$, $P_{\text{all}} := \{0, \ldots, n-1\}$; $g_{\text{LastImp}} := \infty$;
2 repeat
3 $\sigma := P_{\text{best}}$, $g := 0$, $\sigma_{\text{best}} := \sigma$, $P_{\text{all}} := \{0, \ldots, n-1\}$;
4 $P_{\text{base}} := \text{random}(P_{\text{all}})$;
5 $P_{\text{part}} := P_{\text{part}} \setminus \{P_{\text{base}}\}$, $P_{\text{all}} := P_{\text{all}} \setminus \{P_{\text{base}}\}$;
6 repeat
7 find a node $i$ with $\min_{i \in P_{\text{part}}} \delta_i := \text{SwapGain}(i_{\text{base}}, i, \sigma)$;
8 $\sigma := \text{SwapMove}(i_{\text{base}}, i, \sigma)$;
9 $g := g + \delta_i$, $P_{\text{part}} := P_{\text{part}} \setminus \{i\}$;
10 if $g > g_{\text{best}}$ then $\sigma_{\text{best}} := \sigma$, $g_{\text{best}} := g$;
11 until $P_{\text{part}} = \emptyset$ or $g > g_{\text{LastImp}}$;
12 if $g_{\text{best}} < 0$ then $P_{\text{all}} := \{0, \ldots, n-1\}$; $g_{\text{LastImp}} := |g_{\text{best}}|$;
13 until $P_{\text{all}} = \emptyset$;
14 return $\sigma_{\text{best}}$.
end;

---

Fig. 4. Variable k-swap Local Search for NPP

Standard VDS-based local search algorithms applied to several optimization problems so far attempt to select at each iteration the best solution among the chained neighbors that can be obtained by applying a sequence of small move operations based on the best improvement move strategy (see [29], [26], [10] for example). This seems to be impractical for the standard VDS to the NPP because the variable neighborhood size is still large for each iteration. We therefore devise the k-swap neighborhood search in KLS so that the maximum number of chained neighbors $\sigma'$ in the sequence based on 1-swap move operations to $\sigma$ is restricted to $n - 1$ at each iteration instead of $n(n-1)/2$ in the impractical, standard VDS for the NPP. This devised search is achieved with a hybrid scheme based on two types of the move strategies used in each of our 1-swap local search methods.

The pseudo-code of KLS is shown in Figure 4. KLS has outer (lines 2–13) and inner (lines 6–11) loops. They provide the variable k-swap neighborhood search that is the main process of KLS to the current solution $\sigma$, and realize the sequence based on swapping of node pairs selected by the hybrid scheme described above. At lines 1 and 3 $P_{\text{all}}$ and $P_{\text{part}}$ are initialized. They ensure that no node is allowed to be swapped twice in one sequence in order to avoid the cycling among the chained neighbors. In outer loop, we select a random node $i_{\text{base}}$ from $P_{\text{all}}$ at line 4. The selected node $i_{\text{base}}$ is removed from each of $P_{\text{all}}$ and $P_{\text{part}}$ since the node $i_{\text{base}}$ is one of the pair. In inner loop, the counterpart node $i$ is selected from all candidates $i \in P_{\text{part}}$ at line 7 such that the gain value of SwapGain($i_{\text{base}}, i, \sigma$) is minimal even if the gain value is larger than zero. The pair of nodes $i_{\text{base}}$ and the counterpart $i$ selected is swapped at line 8 to move to a neighbor solution from $\sigma$. At the line 9, the gain $g$ is updated, and the node $i$ is removed from $P_{\text{part}}$. If the gain $g$ is smaller than the gain of the best solution found so far in the search, the best solution $\sigma_{\text{best}}$ and the corresponding best gain value $g_{\text{best}}$ are saved. The search of inner loop is repeated until $P_{\text{part}} = \emptyset$. If the best gain value is better than zero at line 12, $P_{\text{all}}$ is initialized, and the k-swap neighborhood search is repeated until $P_{\text{all}} = \emptyset$. Finally, the best solution $\sigma_{\text{best}}$ found in the search is returned.
desire in designing a kick method for a specific problem is to reduce the probability to fall back into a previously found local optimum without moving to a search point where is far away from the current one.

Let us consider a search space in optimization problems briefly. ILS (or MA) should be applied to a problem characterized by the following assumption (or fact): “there are better (local optimum) solutions around the other good one in the search space”. This assumption implies that many local optima are distributed in a cluster so as to lead toward the global optima in the search space. For such optimization problems in which globally convex or big valley structure has been revealed, e.g., TSP, GPP [32] and UBQP [28], it is quite expected that ILS is more favorable in terms of final solution qualities and running times than MLS. The NPP in this paper also is considered as one of such problems.

B. Kick Method and ILS for NPP

One of the simplest kick techniques for the NPP is to randomly select \( \alpha \) nodes from \( n \) nodes in a given solution, and to randomly shuffle the \( \alpha \) nodes to create a kicked solution. However, this kick has the following drawbacks: 1) it is difficult to determine the number of \( \alpha \) that corresponds to perturbation strength [14] because a suitable \( \alpha \) may depend on the instance sizes, solutions given, etc. 2) the positions of slots in which the nodes are assigned in the graph are not taken into account although the number of hops depends on the relation between the slot positions and assigned nodes.

To overcome the potential drawbacks, we propose a kick (mutation) technique, called Cross-Kick, for ILS to the NPP. Given a solution \( \sigma \), Cross-Kick first selects a single node \( i \) randomly from the nodes that cause the highest number of hops in \( \sigma \). We here assume that the node \( i \) selected is assigned at slot \((x, y)\) in the graph, where \( 0 \leq x, y \leq m - 1 \). It next randomly shuffles the \( 2m - 1 \) nodes assigned to the horizontal \((x, j)\) slots and the vertical \((j, y)\) slots based on the node \( i \) assigned at the slot \((x, y)\) that is the cross point, where \( j = \{0, 1, \ldots, m - 1\} \), and a kicked solution is produced. Note that the solutions produced by Cross-Kick can be reached by swapping node pair \( 2m - 2 \) times in the worst case on the horizontal and vertical slots selected in a given solution.

Our framework of ILS for the NPP is quite simple as shown in Figure 5. This simple framework delivers to minimize the number of parameter setting values by user. Only one parameter is required in our ILS. The parameter is to stop ILS at line 8. Note that no parameter setting is required in the kick and local search processes either.

In ILS, an initial solution \( \sigma \) is generated randomly at line 1, and is locally optimized by local search. The resulting solution obtained is saved as the best found solution \( \sigma_{best} \). Lines 3–8 is the main process of ILS, in which Cross-Kick described above is applied to the best solution found during ILS, and the kicked solution is locally optimized by local search to obtain a local optimum \( \sigma \). At line 6, the costs of \( \sigma \) and \( \sigma_{best} \) are compared, and the best found solution becomes the solution for the next iteration to repeat the main process of ILS. The process of ILS is repeated until the termination condition is satisfied. We adopt the execution number of local searches in ILS as the condition in our experiments.

At lines 2 and 5 in Figure 5 we can adopt the 1-swap local search and the \( k \)-swap local search. Although the existing metaheuristics such as the simulated annealing [3] and the tabu search [2] also can be used, it is not easy to set the parameter values in advance that should be used to stop the metaheuristic searches or to change to the kick process during ILS. These parameter settings are quite troublesome, whereas we do not require such parameter settings beforehand for our local search methods because their searches stop when local optima are found in the algorithms.

If we adopt the \( k \)-swap local search (KLS) in the local search process of ILS, we denote Iterated KLS (IKLS). In other cases of 1-swap local search methods, we name ILS-BI and ILS-FI that are based on each of the move strategies: best improvement and first improvement, respectively.

V. EXPERIMENTAL RESULTS

To evaluate the performance of the 1-swap and \( k \)-swap local search methods and their extensions to ILS, we performed extensive experiments on the random instances provided as the NPP benchmarks. We then showed the effectiveness and efficiency of iterated \( k \)-swap local search (IKLS) through comparisons with multistart methods and the published results of the best available metaheuristic for the NPP.

A. Generation of Random Instances with Known-Optimum

In the problem definition for the NPP, each of \( n \) node slots has four bidirectional edges in the network graph. It means that each has four outgoing links. The total number of outgoing links in the graph is equal to \( 4 \times n \). Since the traffic amounts of the outgoing links are represented by an \( n \times n \) traffic matrix \( T = (t_{ij}) \), the random instances of the size \( n \) (\( m \times m \)) can be generated by assigning each of two traffic flows (heavy: \( t_{ii} = 1 \) or light: \( t_{ii} = 0 \)) randomly to each of every outgoing links. If an arbitrary node placement \( \sigma^* \) in the graph, e.g., \( \sigma^* = \{0, 1, 2, \ldots, n - 1\} \), is given as the optimal solution beforehand, we can generate the random instances for which the known-optimum solution \( \sigma^* \) can be specified because all heavy traffic flows are assigned in adjacent nodes to be of single hop. We generated the random instances according to the same manner shown in [2], [3] because the instances used in [2], [3] are not available.
Given parameters $a$ ($0 < a < 1$) and $L_{\text{max}}$ that denotes the maximum number of the links on which heavy traffic flows are given per node ($1 \leq L_{\text{max}} \leq 4$), we randomly select $[4 \times a \times n]$ links from the $4 \times n$ outgoing links, and assign heavy traffic flows ($t_H = 1$) to the links selected randomly so that $L_{\text{max}}$ is satisfied. For the remaining links, light traffic flows ($t_L = 0$) are assigned to make a traffic matrix.

For the experiments, we generated four different sets of the random instances that are identified by the following problem sizes: $n = 16$ ($=4 \times 4$), $64$ ($=8 \times 8$), $256$ ($=16 \times 16$), $1024$ ($=32 \times 32$). Each set consists of 20 different traffic matrices generated with the following parameter values: $a = 0.3$ and $L_{\text{max}} = 3$. Therefore, the total number of the instances with known-optimum are 80. These problem sizes and parameter values are the same in [3]. The optimal solution values $f(\sigma^*)$ are 19, 76, 307, and 1228 for all 20 instances in each set.

For future studies, the instances are available from our web page http://k2x.ice.ous.ac.jp/~katayama/bench/npp/ as the benchmark problem instances for the NPP.

### B. Results for Local Search and Iterated Local Search

All algorithm codes were written in C, and all experiments were performed on Hewlett-Packard xw4300 workstation with Pentium 3.4GHz, 4GB RAM, and Fedora Core 5, using the gcc compiler 4.11 with `-O3` option.

To see the performances of three local search methods shown in this paper: Best Improvement 1-swap Local Search (1LS-BI), First Improvement 1-swap Local Search (1LS-FI), and variable $k$-swap Local Search (KLS), we tested each local search that starts with a randomly generated solution in a single run for each of 20 instances in each problem set. Table I shows the results of the methods. The first two columns are the local search names and the size $n$ of each problem set. In the following columns we show the average cost value $f(\sigma)$ of the 20 solutions obtained, its quality “$Q$” $^1$, and the average running time “Time(s)” $^2$ in seconds.

In comparison between the 1-swap local search methods, 1LS-BI and 1LS-FI, better solutions can be obtained by 1LS-FI with shorter running times in many cases. The results of KLS are further better than those of the others although the running times are larger.

Additional experimental results on the largest problem set $n = 1024$ showed that the average solution quality and the average running time of the First Improvement 1-swap Local Search without don’t look bit (1LS-FI-woDLS) were 0.3272 and 1.49 seconds, resp., and the quality and the time of variable $k$-swap Local Search without the modification for the termination condition of inner loop (KLS-woMod) were 0.4834 and 574.18 seconds, resp. In comparison to the corresponding results shown in Table I, 1LS-FI is faster than 1LS-FI-woDLS about 5.32 times without large loss of final solution quality, and KLS is faster than KLS-woMod about 75 times. These results indicate that the technique of DLB is useful for 1LS-FI, and the modification in KLS contributes to increase the efficiency without loss of solution quality.

Table II displays the results of the multistart methods: multistart 1LS-BI (M1LS-BI), multistart 1LS-FI (M1LS-FI), and multistart KLS (MKLS). The iteration number of each method, i.e., the execution number of local searches, is set to $n$ ($= m \times m$). The columns “$f$”, “$Q$”, “Time(s)”, and “#Iter” for each problem set in Table II showed the average cost value of the best solutions obtained in each of 20 instances, its quality, the average running time in which the algorithm found the best solution during the search, and the average number of local searches to the best solution, resp. It is observed from Table II that the performance of each local search can be improved slightly by adopting the multistart framework. M1LS-FI was the fastest algorithm and obtained competitive solutions in comparison to M1LS-BI. The solution qualities obtained by MKLS were better than the others. We therefore focus on the fastest local search 1LS-FI and the effective KLS in the following comparison.

### TABLE I

<table>
<thead>
<tr>
<th>Method</th>
<th>$n$</th>
<th>$f$</th>
<th>$Q$</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1LS-BI</td>
<td>16</td>
<td>22.50</td>
<td>0.8338</td>
<td>&lt; $\epsilon$</td>
</tr>
<tr>
<td>256</td>
<td>4027.15</td>
<td>0.3049</td>
<td>2.75</td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>118.15</td>
<td>0.6433</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1LS-FI</td>
<td>16</td>
<td>22.50</td>
<td>0.8338</td>
<td>&lt; $\epsilon$</td>
</tr>
<tr>
<td>256</td>
<td>118.15</td>
<td>0.6433</td>
<td>&lt; $\epsilon$</td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>22.50</td>
<td>0.8444</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KLS</td>
<td>16</td>
<td>20.70</td>
<td>0.9179</td>
<td>&lt; $\epsilon$</td>
</tr>
<tr>
<td>64</td>
<td>95.75</td>
<td>0.7937</td>
<td>&lt; $\epsilon$</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>475.20</td>
<td>0.6460</td>
<td>0.12</td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>2592.75</td>
<td>0.4736</td>
<td>7.66</td>
<td></td>
</tr>
</tbody>
</table>

### TABLE II

<table>
<thead>
<tr>
<th>Method</th>
<th>$n$</th>
<th>$f$</th>
<th>$Q$</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1LS-BI</td>
<td>16</td>
<td>19.75</td>
<td>0.9620</td>
<td>&lt; $\epsilon$</td>
</tr>
<tr>
<td>256</td>
<td>582.50</td>
<td>0.5270</td>
<td>5.71</td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>6844.40</td>
<td>0.3333</td>
<td>1609.44</td>
<td></td>
</tr>
<tr>
<td>1LS-FI</td>
<td>16</td>
<td>20.25</td>
<td>0.9383</td>
<td>&lt; $\epsilon$</td>
</tr>
<tr>
<td>256</td>
<td>103.45</td>
<td>0.7347</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>577.45</td>
<td>0.5316</td>
<td>0.95</td>
<td></td>
</tr>
<tr>
<td>KLS</td>
<td>16</td>
<td>19.00</td>
<td>1.0000</td>
<td>&lt; $\epsilon$</td>
</tr>
<tr>
<td>64</td>
<td>86.95</td>
<td>0.8741</td>
<td>0.12</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>423.95</td>
<td>0.7241</td>
<td>15.69</td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>2316.30</td>
<td>0.5302</td>
<td>4086.44</td>
<td></td>
</tr>
</tbody>
</table>

### TABLE III

<table>
<thead>
<tr>
<th>Method</th>
<th>$n$</th>
<th>$f$</th>
<th>$Q$</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1LS-FI</td>
<td>16</td>
<td>19.90</td>
<td>0.9548</td>
<td>&lt; $\epsilon$</td>
</tr>
<tr>
<td>256</td>
<td>94.90</td>
<td>0.8008</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>467.20</td>
<td>0.6571</td>
<td>1.04</td>
<td></td>
</tr>
<tr>
<td>KLS</td>
<td>16</td>
<td>19.00</td>
<td>1.0000</td>
<td>&lt; $\epsilon$</td>
</tr>
<tr>
<td>64</td>
<td>82.25</td>
<td>0.9240</td>
<td>0.14</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>366.20</td>
<td>0.8383</td>
<td>15.92</td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>1681.55</td>
<td>0.7303</td>
<td>3230.30</td>
<td></td>
</tr>
</tbody>
</table>

$^1$ $Q$ can be calculated by $Q = f(\sigma^*)/f(\sigma)$, where $\sigma^*$ is the optimum solution. It therefore that the obtained $\sigma$ is the optimum if $Q = 1.0$.

$^2$ In all tables in this paper, the average running times less than 0.01 seconds are shown as “< $\epsilon$”.

---

2007 IEEE Congress on Evolutionary Computation (CEC 2007) 3513
Table III shows the results of the iterated local search methods: iterated ILS-FI (IILS-FI) and iterated KLS (IKLS). The termination condition of each method is the same setting with the multistart methods shown above. It is observed that the performance of each local search is further improved by adopting the iterated framework. In the results on larger problem sets, IILS-FI obtained considerably better solutions than I1LS-FI in spite of almost the same average running times in both algorithms although the number of “Iter” in I1LS-FI was larger than the multistart case. The similar observations can be found in the results of IKLS and MKLS for all problem sets except for the smallest one. It therefore is considered that the iterated framework is more suitable than the multistart one for the NPP, and it suggests that the search space in the NPP is the big valley structure [32].

Since the results shown in Tables II and III are ones in the final search stage of the multistart and iterated local search methods, it is difficult to observe the performance difference in earlier stages between the methods. Figure 6 provides that the search tendency between solution cost and running time (until 180 seconds) for each method. The figure clearly shows that IKLS is capable of finding better solutions with shorter running times than the others in earlier search stages.

C. Comparison with State-of-the-art Metaheuristics

We next compared the performance of IKLS with that of the currently best metaheuristic for solving the NPP: the simulated annealing, called HIWAS [3]. The procedure HIWAS has been written by C, and the results have been obtained on Pentium-III 1 GHz with 768 MB RAM.

It is difficult to strictly compare the results and efficiency of IKLS with those of HIWAS reported, due to different computers, OS, problem instances (in the strict sense), etc. Since both of HIWAS and IKLS are metaheuristic algorithms, better solutions can be obtained if we permit longer running times for the algorithms. However, it is possible to make a rough comparison for the algorithm efficiencies if we deal with the same period of a specified cost value or quality of obtainable solutions. From this point of view, one of the most suitable measures is to compare the running times and evaluation numbers (move times) to reach the exactly same cost of solutions obtained by the algorithms if both are capable of finding almost the same high-quality solutions.

Table IV summarizes the average results of solution cost \( f \) and running time of HIWAS that are the best result for each problem set reported in [3] and the running time of IKLS to reach the same average solution costs reported in [3]. For example, in the largest problem set \( n = 1024 \), HIWAS took about 2000 seconds on their computer to obtain the solutions of cost 2139.8 on average, whereas IKLS spent about 70 seconds on our computer to reach the same solution quality. The column “\( Q(f) \)” in Table V stands for the specified solution quality investigated in [3] and its corresponding solution cost for each problem set. Tables IV and V suggest that the running times of IKLS are shorter than those of HIWAS for larger problem sets to reach the same solution costs even if we roughly estimated that our computer would be better than one used in [3] ten times.

To further investigate and emphasize the efficiency of our IKLS, we show the number of 1-swap move times performed in IKLS to reach the specified solution quality, as Kitani, et al. [3] also have reported the evaluation number of the cost function \( f \) in HIWAS for each problem set. The column “\( Q(f) \)” in Table VI is the same shown in Table V. The column “IKLS” consists of three sub-columns: the average numbers of 1-swap move times applied in KLS that corresponds to the evaluation numbers investigated in [3], the average move times based on 1-swap move operation in Cross-Kick that can be counted up to \( 2m - 2 \) (see the subsection IV-B) per one kick (in each iteration of IKLS), and the total numbers.

![Graph showing comparison results of solution cost and running time](image)

**Table IV**

<table>
<thead>
<tr>
<th>( n )</th>
<th>( f )</th>
<th>IKLS</th>
<th>HIWAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>197.9</td>
<td>&lt; ( \epsilon )</td>
<td>1024.6</td>
</tr>
<tr>
<td>64</td>
<td>97.6</td>
<td>( &lt; \epsilon )</td>
<td>1.133</td>
</tr>
<tr>
<td>256</td>
<td>460.8</td>
<td>0.25</td>
<td>42.95</td>
</tr>
<tr>
<td>1024</td>
<td>2139.8</td>
<td>69.56</td>
<td>2046.2</td>
</tr>
</tbody>
</table>

**Table V**

<table>
<thead>
<tr>
<th>( n )</th>
<th>( Q(f) )</th>
<th>IKLS</th>
<th>HIWAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>1.9 (19.00)</td>
<td>&lt; ( \epsilon )</td>
<td>1.026</td>
</tr>
<tr>
<td>64</td>
<td>0.8 (95.00)</td>
<td>&lt; ( \epsilon )</td>
<td>2.232</td>
</tr>
<tr>
<td>256</td>
<td>0.7 (438.57)</td>
<td>0.52</td>
<td>85.48</td>
</tr>
<tr>
<td>1024</td>
<td>0.6 (2046.67)</td>
<td>98.74</td>
<td>6080.6</td>
</tr>
</tbody>
</table>

**Table VI**

<table>
<thead>
<tr>
<th>( n )</th>
<th>( Q(f) )</th>
<th>IKLS</th>
<th>HIWAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>1.9 (19.00)</td>
<td>&lt; ( \epsilon )</td>
<td>1.026</td>
</tr>
<tr>
<td>64</td>
<td>0.8 (95.00)</td>
<td>&lt; ( \epsilon )</td>
<td>2.232</td>
</tr>
<tr>
<td>256</td>
<td>0.7 (438.57)</td>
<td>0.52</td>
<td>85.48</td>
</tr>
<tr>
<td>1024</td>
<td>0.6 (2046.67)</td>
<td>98.74</td>
<td>6080.6</td>
</tr>
</tbody>
</table>
of the move times in KLS and Cross-Kick during IKLS to reach the specified solution quality for each problem set. The final column shows the evaluation numbers reported in [3]. It clearly indicates that the move times in IKLS is less than those in HIWAS, and IKLS is more efficient than HIWAS.

We conclude that through the above comparisons, IKLS is superior to the state-of-the-art metaheuristic HIWAS in terms of running times and evaluation (move) times to find the same quality of solutions for the NPP.

VI. CONCLUSION

We addressed the node placement problem (NPP) in multihop WDM lightwave network (BMSN), and proposed the variable k-swap local search (KLS) based on variable depth search and its extension to an iterated local search framework with Cross-Kick. Computational results showed that the iterated local search embedded with KLS, called IKLS, is more effective than the multistart local search methods tested, and outperforms the simulated annealing that is known to be the best available metaheuristic for the NPP.

To evaluate the algorithm performances fairly in future studies, we provided the benchmark instances with known optimal solutions for the NPP. More work remains to be carried out in investigating a wider class of problem instances and revealing clearly the search space structure. Other important works are to investigate the search property of KLS in detail, and to develop and evaluate new powerful metaheuristics for solving the NPP through comparisons with the newly proposed best metaheuristic IKLS.

ACKNOWLEDGMENTS

This research was partly supported by the Grant-in-Aid for Scientific Research (C), no. 19500200, Japan Society for the Promotion of Science.

REFERENCES