Composite Neighborhood Search with a Case Study on Incremental Graph Drawing

Zhouxing Su[1] , **Zhipeng Lü**[1*] , **Chumin Li**[2] , **Qi Jin**[1] and **Bo Peng**[3]
[1]SMART, Huazhong University of Science and Technology, China
[2]MIS, Université de Picardie Jules Verne, France
[3]Southwestern University of Finance and Economics, China
zhipeng.lv@hust.edu.cn

Abstract

Traditional local search algorithms usually evaluate the whole neighborhood but only perform one single move at each step. However, there are often many neighborhood moves in the current neighborhood that are independent of each other and can be simultaneously performed without interference. In this paper, we present a composite neighborhood search (CNS) framework for local search algorithms allowing to perform the best combination of independent moves at each step, by solving the incremental graph drawing problem (IGDP) as a case study. We show that finding the best combination of independent moves can be formulated as the maximum weight clique problem in the general case, and for the IGDP, the best combination of independent moves can be more easily identified using problem-specific knowledge. We test the CNS framework on 240 public IGDP instances, and show its clear advantage in terms of both efficiency and effectiveness. We also carry out an analysis of why and how the CNS framework performs better than traditional single-move-based local search.

1 Introduction

Neighborhood search or local search is known to be a highly effective metaheuristic framework for solving a large number of combinatorial optimization problems. Starting from an initial solution, local search iteratively improves the incumbent solution by exploring its neighborhoods. In this way, the quality of the current solution is progressively improved by one of its neighbors until a specific stop criterion is met.

One of the most important features in a local search algorithm is the definition of its neighborhood. In general, the behavior of local search depends strongly on the characteristics of its neighborhood. Many algorithms employ simple neighborhood moves, such as one flip (move) or two swap, which searches a small neighborhood structure in a fast manner but can only enable small improvements at each step, while other algorithms employ large and complex neighborhood moves, such as ejection chain [Yagiura *et al.*, 2004] or large neighborhood search [Pisinger and Ropke, 2010], which allows the search to obtain solution improvements in a quick and important manner, but the size of the neighborhood structure is huge and the evaluation of the neighborhood is quite expensive.

[*]Contact Author

All the above-mentioned local search algorithms evaluate the whole neighborhood, but only perform one move at each step of the neighborhood search. One could ask the following questions: Why not to simultaneously perform multiple neighborhood moves at each step to improve the current solution as far as possible after evaluating the current neighborhood? How is this kind of combination of neighborhood moves defined and how can the best combination of moves be identified in an effective and efficient manner? How can this approach affect the performance of the traditional local search algorithm?

In this paper, we try to answer these questions by proposing and studying a composite neighborhood search (CNS) framework for local search algorithms. Concretely, we introduce the notion of independent moves in a neighborhood, and show that finding the best combination of independent moves can be formulated and solved as the maximum weight clique problem (MWCP) in the general case. Using this framework, a local search algorithm can simultaneously perform the combination of independent moves bringing the greatest improvement at each step of the local search, which is better than performing only one single move. As a case study, we consider the so-called multi-layered incremental graph drawing problem, one of the hot topics of data visualization nowadays [Beck *et al.*, 2017]. We show that the optimal combination of independent neighborhood moves can be more efficiently identified using dynamic programming, because problem-specific knowledge can be used. The computational results demonstrate that the proposed algorithm is quite competitive compared with the state-of-the-art algorithms in the literature. Furthermore, we carry out an analysis to give insight on why and how composite neighborhood search can be used in local search algorithms for solving combinatorial optimization problems.

2 Motivations and Contributions

As is known, the most time-consuming part of a local search algorithm is the evaluation of the neighborhood. However, when thousands or even millions of (simple or complex) neighborhood moves are evaluated, usually only one move is selected and performed. This leads to the low efficiency of the local search algorithm when we consider the number of performed moves with respect to the evaluated ones. However, there are often many neighborhood moves that are independent of each other and can be performed simultaneously without interference to increase the search efficiency.

More importantly, simultaneously performing a set of neighborhood moves starting from the current solution could allow a more diversified search than successively performing the best move, because independent moves are generally widely distributed in the neighborhood, while successively performing the best move could more easily lead the search to a greedy local suboptimal solution.

All these aspects motivate us to study the composite neighborhood search framework. The contributions of this paper can be summarized as follows:

1) We present the CNS framework for all combinatorial optimization problems, including the definition of independent moves in a neighborhood and the formulation of the problem of finding the best combination of independent moves as MWCP in the general case.
2) Using the incremental graph drawing problem (IGDP) as a case study, we show how moves can be independent and propose a dynamic programming method to identify the optimal combination of independent moves, allowing us to implement CNS algorithms for the IGDP.
3) Tested on 240 public IGDP instances, our CNS algorithms obtain highly competitive results comparing to the state-of-the-art algorithms in the literature.
4) We show that even using an exact algorithm of the MWCP to find the best combination of independent moves in each step without using problem-specific knowledge, our CNS algorithm remains competitive, suggesting the potential of the CNS framework for other combinatorial optimization problems.

3 Composite Neighborhood Search

We propose a composite neighborhood search framework as shown in Algorithm 1. The CNS framework is similar to the classic iterated local search framework. Starting from an initial solution (line 1), it evaluates all neighborhood moves M for the current solution s iteratively (line 5), and escapes the local optima using a perturbation operator (line 9). However, instead of making the best neighborhood move, it identifies and performs a set of independent moves M^* to maximize the total improvement to the objective value (lines 5-7).

In order to fully exploit the potential of the evaluated moves, the CNS framework constructs a composite neighborhood move to simultaneously perform multiple independent moves. Given a feasible solution X, let $f(X)$ be the objective value of X, let binary operator $\oplus$ represent the operation of making a move, let $\Delta(m) = f(X \oplus m) - f(X)$ be the objective improvement of a neighborhood move m, we define the independence of a set of neighborhood moves as follows.

Definition 1. *Neighborhood moves m_1, m_2, ..., m_k to be performed on the incumbent solution X are independent if* $f(X \oplus m_1 \oplus m_2 \oplus \cdots \oplus m_k) = f(X) + \sum_{i=1}^{k} \Delta(m_i)$.

Algorithm 1 Composite Neighborhood Search Framework

```
Input: Instance $I$
Output: Best solution $s^*$ found
 1: $s \leftarrow$ GenerateInitialSolution() // $s$ is the current solution
 2: while Termination condition is not met do
 3:     repeat
 4:         $s^* \leftarrow s$
 5:         $M \leftarrow$ EvaluateNeighborhoodMoves($s$)
 6:         $M^* \leftarrow$ AssembleIndependentMoves($s$, $M$)
 7:         $s \leftarrow$ MakeMoves($s$, $M^*$)
 8:     until $s$ is worse than $s^*$
 9:     $S \leftarrow$ Perturb($s$, $s^*$)
10: end while
```

Definition 1 suggests that the objective improvement of a set of independent moves can be calculated by simply summing up their objective improvements with respect to the current solution. So, instead of making the moves one by one and carefully considering the consequences of each move and the intermediate solutions, one can easily choose a set of independent moves to make simultaneously based on their individual objective improvements. In practice, pairwise independence is much easier to identify, so we use the following assumption to simplify the implementation of CNS.

Hypothesis 1. *Pairwise independence $f(X \oplus m_i \oplus m_j) = f(X)+\Delta(m_i)+\Delta(m_j), \forall i \neq j, 1 \leq i, j \leq k$ derives $f(X \oplus m_1 \oplus m_2 \oplus \cdots \oplus m_k) = f(X) + \sum_{i=1}^{k} \Delta(m_i)$.*

Hypothesis 1 is reasonable because independent moves usually operate on different parts of X. For example, independent neighborhood moves can be easily identified in k-coloring problem and k-set covering problem as follows:

- ***k*-Coloring.** A classic neighborhood move $m(v, c', c)$ for the k-coloring problem can be defined as changing color c' of a vertex v to c where $c \neq c'$. Two moves $m(v_1, c'_1, c_1)$ and $m(v_2, c'_2, c_2)$ are independent if vertex v_1 is not adjacent to vertex v_2.
- ***k*-Set covering.** The unicost k-set covering problem seeks for the full coverage to all elements with exact k sets. The basic neighborhood move $m(s)$ is to flip the selection state of a set s. As long as the flipped sets s and s' do not intersect, the corresponding moves $m(s)$ and $m(s')$ are independent to each other.

Hypothesis 1 is apparently true for these two problems.

Based on this hypothesis, one may observe that finding the best combination of independent moves in the neighborhood of the current solution can be regarded as a maximum weight clique problem (MWCP). Let the moves be nodes in a graph and let $\Delta(m)$ be the weight of the node corresponding to move m. For each pair of independent neighborhood moves, an edge is added between them. Then, the maximum weight clique in the transformed graph gives the optimal combination of independent moves. Generally, one can utilize the state-of-the-art algorithms to solve it exactly [Jiang *et al.*, 2017; Fang *et al.*, 2016] or heuristically [Wang *et al.*, 2016; Wu *et al.*, 2012]. In some special cases where the structure of the resulting graph meet certain criteria, some polynomial algorithms like Balas and Yu; Brandstädt and Mosca [1989; 2018] can be employed to find the optimal combination efficiently. Moreover, dedicated algorithms utilizing problem-specific knowledge can be developed to achieve even better performance, as presented in the following case study.

4 Incremental Graph Drawing Problem

Graph is a powerful model which has brought significant productivity gain in project management [Burch *et al.*, 2012], production planning [Eppler and Platts, 2009], network design [Kriegel *et al.*, 2008], decision making [Mateescu *et al.*, 2008] and data visualization [Hu *et al.*, 2016]. In order to make the graphs more user-friendly, the graph drawing algorithms usually need to place the vertices in a structured layout and minimize the number of crossing edges. One of the most common layouts is the layered graph. A layered graph or hierarchical graph is a topological graph in which the vertices can be partitioned into a set of layers and the edges connect vertices from different layers. As the network grows, additional vertices representing new entities can be added to the original graph constantly. However, users are familiar with the original graph and keeps a so-called mental map in their minds, so disrupting the relative positions of the original vertices should be avoided [Branke, 2001; Görg *et al.*, 2005]. The incremental layered graph drawing problem (IGDP) aims to handle this situation.

Originally introduced by Sugiyama *et al.* [1981], the layered graph drawing problem was widely studied [Laguna *et al.*, 1997; Pupyrev *et al.*, 2011], and softwares which integrate Sugiyama's heuristic such as UML workbench [Seemann, 1997], Graphviz [Ellson *et al.*, 2004], and GLEE [Nachmanson *et al.*, 2008] have been developed. Apart from the static version, Beck *et al.* [2014] presented comprehensive surveys on a series of dynamic graph drawing problems. However, most early works were based on case studies and there were no objective benchmarks for testing their algorithms. Recently, Sánchez-Oro *et al.* [2017] proposed a variable neighborhood scatter search (VNSS) algorithm to solve the incremental layered graph drawing problem and tested it on 240 instances in IGDPLIB [Laguna *et al.*, 1997]. Napoletano *et al.* [2019] imposed additional constraints to absolute positions of the original vertices, and proposed a GRASP heuristic for solving it. Besides, Martí *et al.* [2018] focused on the 2-layer graphs, and presented a tabu search algorithm for the dynamic bipartite drawing problem.

Given an incremental layered graph $G = (V, E, L)$ where V is the set of vertices, E is the set of edges and L is the set of layers. Let V_l be the set of vertices in layer l, and we define V_l^0 and V_l^+ be the original vertices and the newly added vertices in layer l, respectively. Similarly, we define E_l be the set of edges between layers l and $l+1$, and we can add a dummy $E_{|L|} = \varnothing$ for consistency. It is obvious that $V = V_1 \cup V_2 \cup ... \cup V_{|L|}$ and $E = E_1 \cup E_2 \cup ... \cup E_{|L|}$. Figure 1 illustrates an instance of the IGDP and one of its feasible solutions. There are 6 layers in Figure 1, each of which contains a column of vertices. The black nodes are the original nodes whose relative positions must be preserved,

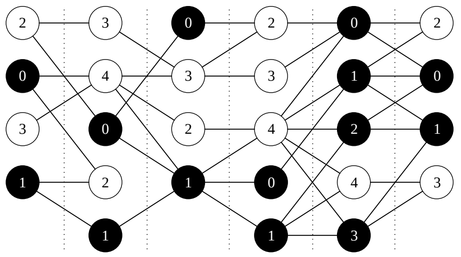


Figure 1: A solution to an incremental layered graph drawing instance. The black nodes are the original nodes and white nodes are the newly added ones.

and the white nodes are the newly added ones which can be arbitrarily placed. For convenience, the partial order on the original nodes is prescribed by the names of the nodes, i.e., node i precedes node j if $i < j$. In the sequel, all figures follow this drawing convention.

In this study, we focus on a special case of the incremental layered graph where edges only exist between adjacent layers. In other words, there is no edge between vertices in the same layer or a pair of non-adjacent layers. Note that the latter case can be handled by adding dummy nodes in each intermediate layer to make the edge a polyline [Rüegg *et al.*, 2016]. Let p_i^l be the position index of vertex i in layer l. Let $x_{ij}^l = 1$ if vertex i precedes vertex j in layer l, i.e., $p_i^l < p_j^l$, otherwise $x_{ij}^l = 0$. Let $c_{ijuv}^l = 1$ if edge (i, u) and edge (j, v) cross, i.e., $(p_i^l - p_j^l)(p_u^{l+1} - p_v^{l+1}) < 0$ or $x_{ij}^l \neq x_{uv}^{l+1}$, where vertices i, j lie in layer l and vertices u, v lie in layer $l+1$, otherwise $c_{ijuv}^l = 0$. The mixed integer programming (MIP) model for the incremental graph drawing is as follows.

$$\min \sum_{l \in L} \sum_{\substack{(i,u),(j,v) \in E_l \\ i<j}} c_{ijuv}^l, \tag{1}$$

$$\text{s.t.} \quad -c_{ijuv}^l \leq x_{ij}^l - x_{uv}^{l+1} \leq c_{ijuv}^l$$
$$\forall l \in L, \forall (i,u),(j,v) \in E_l, i < j, u < v, \tag{2}$$

$$-c_{ijuv}^l \leq x_{ij}^l - (1 - x_{vu}^{l+1}) \leq c_{ijuv}^l$$
$$\forall l \in L, \forall (i,u),(j,v) \in E_l, i < j, u > v, \tag{3}$$

$$1 - |V_l|(1 - x_{ij}^l) \leq p_j^l - p_i^l \leq |V_l| x_{ij}^l - 1$$
$$\forall l \in L, \forall i, j \in V_l, i < j, \tag{4}$$

$$x_{ij}^l = 1, \forall l \in L, \forall i, j \in V_l^0, i < j, \tag{5}$$

$$x_{ij}^l, c_{ijuv}^l \in \{0, 1\}, p_i^l \in [1, |N_l|]$$
$$\forall l \in L, \forall i, j \in N_l, \forall (i,u),(j,v) \in E_l, i < j. \tag{6}$$

The objective is to minimize the total number of crossing edges. Constraints (2) and (3) require that $c_{ijuv}^l \geq x_{ij}^l \oplus x_{uv}^{l+1} = (\neg x_{ij}^l \wedge x_{uv}^{l+1}) \vee (x_{ij}^l \wedge \neg x_{uv}^{l+1})$ so that the crossing state will be correctly counted into the objective. Constraints (4) keep the consistency of the relative position among nodes, that is, if node i precedes node j and node j precedes node k, then node i must precede node k. Constraints (5) impose the relative order of the original nodes.

5 Two CNS Algorithms for IGDP

In order to examine the effectiveness of the CNS framework, we design two algorithms for solving IGDP based on the CNS framework. The first is called CNS-BB and uses a branch-and-bound (BB) algorithm to find the best set of independent moves as maximum weight clique (MWC) without problem-specific knowledge, and the second is called CNS-DP and uses a dedicated dynamic programming algorithm to identify the best set of independent moves for IGDP.

5.1 Neighborhood Structure

We design two kinds of neighborhoods for the IGDP, which are insertion and exchange, respectively. For a pair of nodes i and j in layer l, $\eta^l(i,j)$ represents inserting node i right after node j if node i precedes node j, or inserting node i right before node j if node j precedes node i. Similarly, exchanging the positions of nodes i and j in layer l is denoted by $\chi^l(i,j)$. It is obvious that the exchange operation can be achieved by a pair of insertion operations, so we will only focus on the implementation of the insertion neighborhood. In addition, only one layer is considered at each step, and the layers are picked in turns, i.e., the layer $k \bmod |L|$ will be selected at step k.

For convenience, we define some notations for later uses. Let $\Delta(\eta^l(i,j))$ be the objective improvement of an insertion move, and the objective value of each neighboring solution can be incrementally calculated as $f(X \oplus \eta^l(i,j)) = f(X) + \Delta(\eta^l(i,j))$. Also, we denote the position index of node i in layer l by p_i^l, which is consistent with the MIP model.

5.2 Independent Moves for IGDP

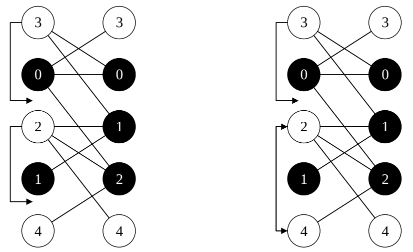
As explained earlier, the problem of identifying the best set of independent moves can be formulated as MWCP. Here we show how to find the best composite move for IGDP.

Proposition 1. *Let a and b be two integers and $[a,b]$ denotes the set of integers $\{x \mid \min\{a,b\} \le x \le \max\{a,b\}\}$. In IGDP, a pair of moves $\eta^l(i,j)$ and $\eta^l(u,v)$ are independent to each other if $[p_i^l,p_j^l] \cap [p_u^l,p_v^l] = \varnothing$.*

Proof. Without loss of generality, we discuss the case $p_i^l < p_j^l < p_u^l < p_v^l$ only. Note that it may introduce non-zero terms (additional or reduced crosses) to the objective improvement $\Delta(\eta^l(i,j))$ and $\Delta(\eta^l(u,v))$ iff the relative positions of any pairs of nodes change. But $\eta^l(i,j)$ ($\eta^l(u,v)$) only disrupts the order of nodes between $[p_i^l,p_j^l]$ ($[p_u^l,p_v^l]$). So, new non-zero terms will never be introduced to $\eta^l(i,j)$ if $\eta^l(u,v)$ is simultaneously performed, and vice versa. Hence, $\Delta(\eta^l(i,j) \oplus \eta^l(u,v)) = \Delta(\eta^l(i,j)) + \Delta(\eta^l(u,v))$. □

Proposition 2. *In IGDP, given a set of neighborhood moves M, if any two moves in M are independent to each other, then M is a set of independent moves.*

Proof. Let $m_1 = \eta^l(i,j) \in M$ and $M' = M \setminus m_1$, it is obvious that if $[p_i^l,p_j^l] \cap [p_u^l,p_v^l] = \varnothing, \forall \eta^l(u,v) \in M'$, then $[p_i^l,p_j^l] \cap (\bigcup_{\eta^l(u,v)\in M'} [p_u^l,p_v^l]) = \varnothing$, which means $f(X \oplus m_1 \oplus m_2 \oplus \cdots \oplus m_k) = f(X \oplus m_2 \oplus \cdots \oplus m_k) + \Delta(m_1)$. It will lead to the equation given by Definition 1 by recursively expanding the right-hand side following the same rule. □



(a) $\eta^0(3,0)$ and $\eta^0(2,1)$. (b) $\eta^0(3,0)$ and $\chi^0(2,4)$.

Figure 2: Different combinations of independent moves. Note that $[p_3^0,p_0^0] \cap [p_2^0,p_1^0] = \varnothing$ and $[p_3^0,p_0^0] \cap [p_2^0,p_4^0] = \varnothing$.

Figure 2 presents an intuitive illustration for the independent moves identified by Proposition 1. From Figure 2 we can observe that an insertion $\eta^l(i,j)$ involves a segment of nodes between nodes i and j. The moves never interfere with each other if the intersection of the involved segments is empty. For example, $\eta^0(3,0)$ is independent with both $\eta^0(2,1)$ and $\chi^0(2,4)$ in Figure 2. So, $\{\eta^0(3,0), \eta^0(2,1)\}$ and $\{\eta^0(3,0), \chi^0(2,4)\}$ are two composite moves as shown in Figures 2a and 2b. Furthermore, it is easy to extend Proposition 1 to multi-move cases as shown in Proposition 2, which means that Hypothesis 1 is true for IGDP.

5.3 Algorithms CNS-BB and CNS-DP for IGDP

CNS-BB is an implementation of Algorithm 1 by implicitly building a graph with moves computed in line 5 (EvaluateNeighborhoodMoves) as vertices, and every edge connects two independent moves. Then, the branch-and-bound MWCP algorithm of Jiang *et al.* [2018] is used to computed an MWC, which is the best set of independent moves, in line 6 (AssembleIndependentMoves).

For IGDP, we can also use problem-specific knowledge to identify the best set of independent moves more efficiently. Let $\Delta^l(p_i^l,p_j^l)$ be the objective improvement of the best neighborhood move between nodes in positions p_i^l and p_j^l, which is calculated by Eq. (7). Note that the two parameters in $\Delta^l(p_i^l,p_j^l)$ are node positions instead of node names, which is different from the previous notations.

$$\Delta^l(p_i^l,p_j^l) = \max\{0, \Delta(\eta^l(i,j)), \Delta(\eta^l(j,i)), \Delta(\chi^l(i,j))\} \quad (7)$$

Let $d^l(k)$ be the objective improvement of the best neighborhood move combination between positions 0 and k in layer l. Then, the recursive formulation for finding the best composite move can be presented as Eq. (8).

$$d^l(k) = \begin{cases} 0 & \text{if } k < 1 \\ \max\limits_{0 \le k' \le k} \{d^l(k'-1) + \Delta^l(k',k)\} & \text{if } k \ge 1 \end{cases} \quad (8)$$

As illustrated in Figure 3, the idea behind Eq. (8) is to decompose $d^l(k)$ into an optimal substructure $d^l(k'-1)$ (before k' part) and a single move between two nodes in positions k' and k (after k' part), and evaluate all splitting points k'. Then, we choose the best splitting point k' to determine $d^l(k)$, which can be done in $O(k)$ time, provided that $d^l(k'-1) + \Delta^l(k',k)$ has been calculated for all $0 \le k' \le k$.

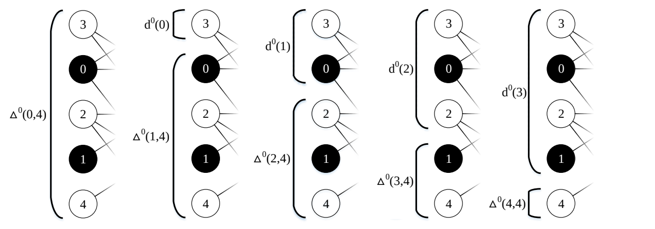


Figure 3: Illustration for the dynamic programming. The optimal $d^0(4)$ will always be the best one among these patterns.

Based on this, we implement an $O(|V_l|^2)$ time dynamic programming procedure to calculate the values from $d^l(0)$ to $d^l(|V_l|-1)$ iteratively, and retrieve the optimal combination of independent moves and its corresponding objective improvement from these values.

CNS-DP is an implementation of Algorithm 1 using the dynamic programming described above to identify the best set of independent moves in line 6.

5.4 Other Components and Strategies

Here we present the remaining components of the CNS algorithms, including the initial solution generation and the perturbation operator. We adopt a simple random initialization for IGDP which arranges the original nodes according to their precedence, and inserts new nodes into random positions. The perturbation operator randomly picks 10% new nodes and inserting them into random positions.

6 Computational Results and Analysis

In order to assess the effectiveness of the CNS framework and its two implementations, we conduct extensive experiments on the 240 IGDP instances presented in Laguna *et al.* [1997] and compare our outcomes with the best results reported by Sánchez-Oro *et al.* [2017]. The testbed is evenly divided into four categories with 2, 6, 13, and 20 layers, respectively. Each category can be further partitioned into three groups whose graph densities are 0.065, 0.175 and 0.300, respectively. Moreover, there are two sub-groups in each group which consists of 10 instances with $0.2|V|$ new nodes and 10 ones with $0.6|V|$ new nodes. The number of vertices in each layer is randomly distributed between 5 and 30. The experiments are run on a server with Intel Xeon E5-2609 v2 2.5GHz CPU and 32GB RAM, and the run time limit is 10 seconds.

Table 1 illustrates the detailed computational results of the proposed algorithm. The first four columns gives the index, layer number, edge density, and new node ratio of each group of instances, respectively. Column VNSS presents the results produced by the variable neighborhood scatter search (VNSS) [Sánchez-Oro *et al.*, 2017]. Columns DP10s and BB10s gives the best results obtained by the CNS-DP and CNS-BB algorithms within a time limit of 10 seconds, respectively. We set a 10-second time limit for these algorithms to test their performance under real-life conditions. Columns MIP1h, DP1h, and BB1h report the best objective values found by solving the proposed MIP model with Gurobi 8.1, CNS-DP, and CNS-BB under one-hour time limit for theoretical analysis. The statistics for each algorithm on each instance are the best results in 10 independent runs, respectively. From Table 1 we can observe that the proposed CNS algorithms dominate VNSS on every instance. It is also highly competitive comparing to the MIP solver, and our advantage gets bigger as the number of new nodes and the edge density increases. Specifically, CNS-DP got the best results on all 240 instances under the one-hour time limit. In contrast, MIP1h fails to match CNS-DP1h on 36 instances, and the average objective gap is over 1%. Apart from the improved objective values, the computational time for convergence is significantly reduced on large-scale instances. CNS-DP is 10 times faster than MIP under the extended time limit, and is 6 times faster than VNSS under the normal time limit. Another interesting fact is that, without using the problem-specific knowledge for finding the best combination of independent moves, CNS-BB still outperforms VNSS within ten seconds and obtains better results on 5 instance groups comparing to MIP1h. In addition, when the time limit is extended, CNS-BB obtains more best known results and better average objective value comparing to MIP and CNS-DP10s, which shows the potential of the CNS framework.

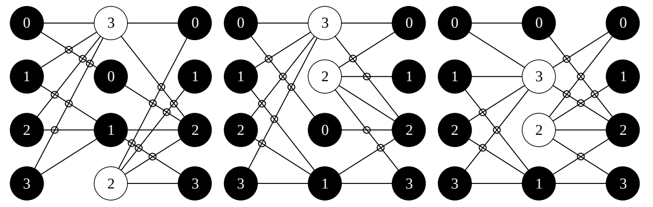

(a) Current solution (13 crosses). (b) Best single move (10 crosses). (c) Best composite move (9 crosses).

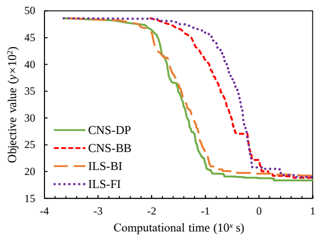
Figure 4: Different local optima of single and composite moves.

The above computational results show great advantage of the CNS framework. Next, we investigate the importance of the composite neighborhood search in our algorithm. The composite neighborhood search framework evaluates neighborhood moves in a thorough and exact way, which is different from the classic best-improvement and first-improvement strategies. Intuitively, this feature could result in quick convergence as the objective improvement per evaluation increases. More importantly, it also improves the intensification of the search and may lead to better local optima under certain circumstances. Figure 4 illustrates different search trajectories of two neighborhood move selection strategies. Given an initial solution in Figure 4a, the best single move is $\eta^1(2,0)$ as shown in Figure 4b. Then it is trapped in the local optimum solution and the search stops. However, the best composite neighborhood move is $\{\eta^1(3,0), \eta^1(2,1)\}$ and it leads to a better local optimum solution as shown in Figure 4c, and it happens to be the global optimum in this tiny instance.

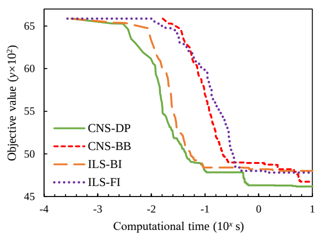
Apart from the intuitive investigation, we conduct experiments to compare the evolution of objective values with the computational time of different strategies on four typical instances, as illustrated in Figure 5. In these figures, CNS-DP and CNS-BB respectively represent our proposed dynamic-programming-based and branch-and-bound-based CNS. ILS-

Table 1: Experimental results on 240 IGDPLIB instances.

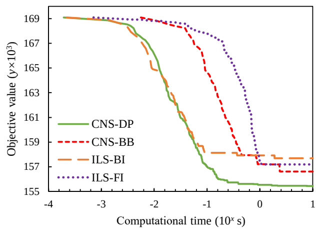
ID	$\|L\|$	Density	$\frac{\|V^+\|}{\|V\|}$	Average best objective value						Average best objective gap (%)					Average CPU (s) for best result					
				VNSS	MIP1h	DP1h	DP10s	BB1h	BB10s	MIP1h	DP1h	DP10s	BB1h	BB10s	VNSS	MIP1h	DP1h	DP10s	BB1h	BB10s
1	2	0.065	0.2	31.4	9.2	9.2	9.2	9.2	9.2	0.00	0.00	0.00	0.00	0.00	0.03	0.14	0.04	0.04	0.03	0.03
2	2	0.065	0.6	29.9	9.9	9.9	9.9	9.9	9.9	0.00	0.00	0.00	0.00	0.00	0.15	0.24	0.08	0.08	0.10	0.10
3	2	0.175	0.2	581.7	520.7	520.7	520.7	520.7	520.7	0.00	0.00	0.00	0.00	0.00	0.04	0.21	0.02	0.02	0.02	0.02
4	2	0.175	0.6	974.0	962.3	962.3	962.3	962.3	962.5	0.00	0.00	0.00	0.00	0.02	0.36	1.08	0.78	0.78	9.63	0.44
5	2	0.300	0.2	2528.9	2440.7	2440.7	2440.7	2440.7	2440.7	0.00	0.00	0.00	0.00	0.00	0.10	0.41	0.02	0.02	0.02	0.02
6	2	0.300	0.6	4369.2	4357.3	4357.3	4357.3	4357.3	4357.3	0.00	0.00	0.00	0.00	0.00	0.63	10.37	0.12	0.12	1.12	1.12
7	6	0.065	0.2	228.4	220.2	220.2	220.2	220.5	220.5	0.00	0.00	0.00	0.43	0.43	0.41	0.44	0.09	0.09	2.47	2.47
8	6	0.065	0.6	469.4	423.5	423.5	431.9	426.7	441.9	0.00	0.00	1.94	1.34	5.09	2.78	7.37	147.05	3.84	359.26	5.51
9	6	0.175	0.2	3113.8	3095.6	3095.6	3095.6	3095.6	3095.6	0.00	0.00	0.00	0.00	0.00	0.66	1.27	0.08	0.08	1.10	1.10
10	6	0.175	0.6	5581.3	5478.0	5478.0	5478.0	5478.1	5482.3	0.00	0.00	0.00	0.00	0.09	5.15	81.94	96.17	2.55	120.15	5.85
11	6	0.300	0.2	11788.7	11633.5	11633.5	11633.5	11633.9	11633.9	0.00	0.00	0.00	0.00	0.00	1.06	2.50	0.05	0.05	0.13	0.13
12	6	0.300	0.6	20867.5	21631.5	20707.2	20707.2	20707.5	20716.6	2.35	0.00	0.00	0.00	0.04	6.88	2180.79	0.99	0.99	16.25	3.91
13	13	0.065	0.2	822.2	793.7	793.7	794.5	794.6	797.4	0.00	0.00	0.10	0.12	0.52	2.05	1.66	141.48	2.35	29.14	4.56
14	13	0.065	0.6	1572.0	1407.1	1407.1	1445.8	1435.8	1485.8	0.00	0.00	2.69	1.96	5.53	16.51	34.28	110.54	2.87	1495.88	4.73
15	13	0.175	0.2	7456.4	7403.1	7403.1	7403.1	7403.1	7406.0	0.00	0.00	0.00	0.00	0.08	3.04	3.64	1.49	1.49	20.53	3.64
16	13	0.175	0.6	13326.9	13404.5	13040.6	13044.5	13061.7	13099.2	1.49	0.00	0.06	0.28	0.68	23.45	1939.75	12.44	4.70	384.59	6.20
17	13	0.300	0.2	26952.2	26888.9	26888.9	26888.9	26888.9	26888.9	0.00	0.00	0.00	0.00	0.00	3.87	6.04	0.12	0.12	1.25	1.25
18	13	0.300	0.6	47795.8	53938.6	47442.4	47442.4	47447.2	47485	11.10	0.00	0.00	0.01	0.12	36.03	3603.91	4.35	4.35	839.09	5.67
19	20	0.065	0.2	1496.1	1465.7	1465.7	1467.5	1465.7	1471.9	0.00	0.00	0.13	0.00	0.42	5.80	3.86	147.38	2.89	334.44	5.92
20	20	0.065	0.6	2845.3	2611.5	2611.5	2687.0	2668.6	2747.2	0.00	0.00	2.97	2.24	5.29	50.05	69.98	366.37	4.41	1137.65	6.99
21	20	0.175	0.2	12272.4	12193.4	12193.4	12193.7	12193.4	12195.6	0.00	0.00	0.00	0.00	0.02	7.78	7.08	27.87	3.20	26.17	4.29
22	20	0.175	0.6	21936.0	22276.6	21465.5	21489.4	21471.1	21610.4	2.06	0.00	0.16	0.04	0.78	73.81	2624.97	208.41	6.08	1656.24	5.47
23	20	0.300	0.2	44720.6	44616.4	44616.4	44616.7	44618.2	44624.5	0.00	0.00	0.00	0.01	0.02	11.21	14.43	39.94	3.02	268.60	3.87
24	20	0.300	0.6	78946.7	91001.2	78266.1	78266.1	78276.1	78346.3	15.91	0.00	0.00	0.02	0.09	83.27	3606.11	3.76	3.76	600.92	6.98
Average				12946.1	13699.3	12810.5	12816.9	12816.1	12835.4	1.37	0.00	0.34	0.27	0.80	14.00	591.77	54.57	2.00	304.37	3.34
#Best				0/240	204/240	240/240	198/240	211/240	148/240											



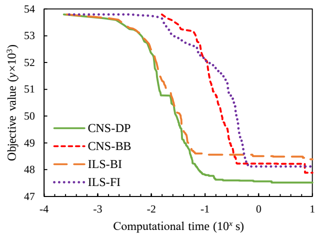
(a) Results on medium instance incgraph_13_0.06_5_30_1.60_2.



(b) Results on medium instance incgraph_13_0.17_5_30_1.60_9.



(c) Results on large instance incgraph_20_0.30_5_30_1.60_1.



(d) Results on large instance incgraph_20_0.30_5_30_1.60_9.

Figure 5: Evolution of objective values with the computational time of different strategies on four typical instances.

BI and ILS-FI denote the classic iterated local search which only performs one single move at each step using the best improvement and the first improvement strategies, respectively.

From Figure 5 we can observe that the results of CNS-DP and ILS-BI are close to each other during the first 0.1 seconds, then CNS-DP quickly converges to much better solutions. When the time limit is reached, CNS-DP keeps dominating other strategies and both implementations of the CNS framework found better solutions comparing to ILS. In particular, the gap between CNS and ILS is significant on the large-scale instances as shown in Figures 5c and 5d. These results indicate that the composite neighborhood structure improves the intensification of the search and make it more powerful.

Regarding the computational efficiency, the x-coordinate of the left-most point of each curve is the time consumed by the first step of local search procedure, which reflects the overhead for the neighborhood evaluation to some extent. As we can see in Figure 5, the branch-and-bound algorithm for the general maximum clique problem is almost 100 times slower than the dedicated dynamic programming algorithm on these instances. Nevertheless, it reaches better local optima than the two ILS algorithms within 10 seconds on these instances. Moreover, the CNS-BB algorithm can continue to improve the solution quality if the time limit is extended, since there is still improvement when it is close to the timeout as shown in Figures 5b and 5d. This demonstrates the potential of the CNS framework to solve other discrete optimization problems even if an efficient algorithm for finding the best combination of independent moves do not exist.

7 Conclusion

We proposed a composite neighborhood search framework which is an iterated local search algorithm integrating a composite neighborhood structure. As a case study, we designed a dynamic-programming-based composite neighborhood search algorithm for solving the incremental graph drawing problem (IGDP) to investigate the performance of the proposed CNS framework. The computational results demonstrate the advantage of the CNS framework for solving this NP-hard problem. Based on these facts, it seems worthy to investigate the efficiency and effectiveness of the CNS framework on other combinatorial optimization problems in the future. In addition, it will be interesting to combine the CNS framework with other successful strategies such as population-based metaheuristics and guided local search.

References

[Balas and Yu, 1989] Egon Balas and Chang Sung Yu. On graphs with polynomially solvable maximum-weight clique problem. *Networks*, 19(2):247–253, 1989.

[Beck *et al.*, 2014] Fabian Beck, Michael Burch, Stephan Diehl, and Daniel Weiskopf. The state of the art in visualizing dynamic graphs. In R. Borgo, R. Maciejewski, and I. Viola, editors, *EuroVis - STARs*. The Eurographics Association, 2014.

[Beck *et al.*, 2017] Fabian Beck, Michael Burch, Stephan Diehl, and Daniel Weiskopf. A taxonomy and survey of dynamic graph visualization. *Computer Graphics Forum*, 36(1):133–159, 2017.

[Brandstädt and Mosca, 2018] Andreas Brandstädt and Raffaele Mosca. Maximum weight independent sets for (p7,triangle)-free graphs in polynomial time. *Discrete Applied Mathematics*, 236:57–65, 2018.

[Branke, 2001] Jürgen Branke. Dynamic graph drawing. In Michael Kaufmann and Dorothea Wagner, editors, *Drawing Graphs: Methods and Models*, pages 228–246. Springer, Berlin, Heidelberg, 2001.

[Burch *et al.*, 2012] Michael Burch, Christoph Müller, Guido Reina, Hansjoerg Schmauder, Miriam Greis, and Daniel Weiskopf. Visualizing dynamic call graphs. In Michael Goesele, Thorsten Grosch, Holger Theisel, Klaus Toennies, and Bernhard Preim, editors, *Vision, Modeling and Visualization*. The Eurographics Association, 2012.

[Ellson *et al.*, 2004] John Ellson, Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Gordon Woodhull. Graphviz and dynagraph — static and dynamic graph drawing tools. In Michael Jünger and Petra Mutzel, editors, *Graph Drawing Software*, pages 127–148. Springer, Berlin, Heidelberg, 2004.

[Eppler and Platts, 2009] Martin J. Eppler and Ken W. Platts. Visual strategizing: The systematic use of visualization in the strategic-planning process. *Long Range Planning*, 42(1):42–74, 2009.

[Fang *et al.*, 2016] Zhiwen Fang, Chu-Min Li, and Ke Xu. An exact algorithm based on maxsat reasoning for the maximum weight clique problem. *Journal of Artificial Intelligence Research*, 55:799–833, 2016.

[Görg *et al.*, 2005] Carsten Görg, Peter Birke, Mathias Pohl, and Stephan Diehl. Dynamic graph drawing of sequences of orthogonal and hierarchical graphs. In János Pach, editor, *Graph Drawing*, pages 228–238, Berlin, Heidelberg, 2005. Springer.

[Hu *et al.*, 2016] Changjun Hu, Yang Li, Xin Cheng, and Zhenyu Liu. A virtual dataspaces model for large-scale materials scientific data access. *Future Generation Computer Systems*, 54:456–468, 2016.

[Jiang *et al.*, 2017] Hua Jiang, Chu-Min Li, and Felip Manya. An exact algorithm for the maximum weight clique problem in large graphs. In *Thirty-First AAAI Conference on Artificial Intelligence*, pages 830–838, 2017.

[Jiang *et al.*, 2018] Hua Jiang, Chu-Min Li, Yanli Liu, and Felip Manya. A two-stage maxsat reasoning approach for the maximum weight clique problem. In *Thirty-Second AAAI Conference on Artificial Intelligence*, pages 1338–1346, 2018.

[Kriegel *et al.*, 2008] Hans-Peter Kriegel, Peer Kröger, Matthias Renz, and Tim Schmidt. Hierarchical graph embedding for efficient query processing in very large traffic networks. In *International Conference on Scientific and Statistical Database Management*, pages 150–167. Springer, 2008.

[Laguna *et al.*, 1997] Manuel Laguna, Rafael Martí, and Vicente Valls. Arc crossing minimization in hierarchical digraphs with tabu search. *Computers & Operations Research*, 24(12):1175–1186, 1997.

[Martí *et al.*, 2018] Rafael Martí, Anna Martínez-Gavara, Jesús Sánchez-Oro, and Abraham Duarte. Tabu search for the dynamic bipartite drawing problem. *Computers & Operations Research*, 91:1–12, 2018.

[Mateescu *et al.*, 2008] Robert Mateescu, Rina Dechter, and Radu Marinescu. AND/OR multi-valued decision diagrams (AOMDDs) for graphical models. *Journal of Artificial Intelligence Research*, 33:465–519, 2008.

[Nachmanson *et al.*, 2008] Lev Nachmanson, George Robertson, and Bongshin Lee. Drawing graphs with GLEE. In Seok-Hee Hong, Takao Nishizeki, and Wu Quan, editors, *Graph Drawing*, pages 389–394, Berlin, Heidelberg, 2008. Springer.

[Napoletano *et al.*, 2019] Antonio Napoletano, Anna Martínez-Gavara, Paola Festa, Tommaso Pastore, and Rafael Martí. Heuristics for the constrained incremental graph drawing problem. *European Journal of Operational Research*, 274(2):710–729, 2019.

[Pisinger and Ropke, 2010] David Pisinger and Stefan Ropke. Large neighborhood search. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, pages 399–419. Springer, Boston, MA, 2010.

[Pupyrev *et al.*, 2011] Sergey Pupyrev, Lev Nachmanson, and Michael Kaufmann. Improving layered graph layouts with edge bundling. In Ulrik Brandes and Sabine Cornelsen, editors, *Graph Drawing*, pages 329–340, Berlin, Heidelberg, 2011. Springer.

[Rüegg *et al.*, 2016] Ulf Rüegg, Thorsten Ehlers, Miro Spönemann, and Reinhard von Hanxleden. A generalization of the directed graph layering problem. In Yifan Hu and Martin Nöllenburg, editors, *Graph Drawing and Network Visualization*, pages 196–208, Cham, 2016. Springer.

[Sánchez-Oro *et al.*, 2017] Jesús Sánchez-Oro, Anna Martínez-Gavara, Manuel Laguna, Rafael Martí, and Abraham Duarte. Variable neighborhood scatter search for the incremental graph drawing problem. *Computational Optimization and Applications*, 68(3):775–797, Dec 2017.

[Seemann, 1997] Jochen Seemann. Extending the Sugiyama algorithm for drawing UML class diagrams: Towards automatic layout of object-oriented software diagrams. In Giuseppe DiBattista, editor, *Graph Drawing*, pages 415–424, Berlin, Heidelberg, 1997. Springer.

[Sugiyama *et al.*, 1981] Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, Feb 1981.

[Wang *et al.*, 2016] Yiyuan Wang, Shaowei Cai, and Minghao Yin. Two efficient local search algorithms for maximum weight clique problem. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 805–811. AAAI Press, 2016.

[Wu *et al.*, 2012] Qinghua Wu, Jin-Kao Hao, and Fred Glover. Multi-neighborhood tabu search for the maximum weight clique problem. *Annals of Operations Research*, 196(1):611–634, Jul 2012.

[Yagiura *et al.*, 2004] Mutsunori Yagiura, Toshihide Ibaraki, and Fred Glover. An ejection chain approach for the generalized assignment problem. *INFORMS Journal on Computing*, 16(2):133–151, 2004.