Very Large-Scale Neighborhood Search: Overview and Case Studies on Coloring Problems

Marco Chiarandini[1], Irina Dumitrescu[2], and Thomas Stützle[3]

[1] Department of Mathematics and Computer Science
University of Southern Denmark, Odense, Denmark
marco@imada.sdu.dk
[2] School of Mathematics and Statistics
University of New South Wales, Sydney, Australia
irina.dumitrescu@unsw.edu.au
[3] Computer & Decision Engineering (CoDE) Department, IRIDIA
Université Libre de Bruxelles, Brussels, Belgium
stuetzle@ulb.ac.be

Summary. Two key issues in local search algorithms are the definition of a neighborhood and the way to examine it. In this chapter we consider techniques for examining very large neighborhoods, in particular, ways for exactly searching them. We first illustrate such techniques using three paradigmatic examples. In the largest part of the chapter, we focus on the development and experimental study of very large-scale neighborhood search algorithms for two coloring problems. The first example concerns the well-known (vertex) graph coloring problem. Despite initial promising results on the use of very large-scale neighborhoods, our final conclusion was negative: the usage of the proposed very large-scale neighborhoods did not help to improve the performance of effective stochastic local search algorithms. The second example, the graph set T-coloring problem, yielded more positive results. In this case, a very large-scale neighborhood that was specially tailored for this problem and that can be efficiently searched, resulted to be an essential component of a new state-of-the-art algorithm for various instance classes.

1 Introduction

Many efficient algorithms for combinatorial optimization problems rely on (perturbative) local search techniques [1, 37]. These start from some initial candidate solution for the problem to be solved and then iteratively replace the current candidate solution by some neighboring one. The neighborhood of a solution comprises all those candidate solutions that are reachable in one logical step of the algorithm. More formally, a neighborhood N can be defined as a function $N : S \to 2^S$ that assigns to every candidate solution s

M. Chiarandini et al.: *Very Large-Scale Neighborhood Search: Overview and Case Studies on Coloring Problems*, Studies in Computational Intelligence (SCI) **114**, 117–150 (2008)
www.springerlink.com

in the search space S a set of neighbors $N(s) \subseteq S$. The simplest local search algorithm only accepts better neighbors and it is therefore called iterative improvement. It terminates once no improving neighbor is available anymore. In this case, a local optimum has been reached.

Often, the neighborhood of a solution s is defined by considering all possible modifications that can be applied to a candidate solution in order to yield a new, different solution. In this case, the neighborhood of s is the set of candidate solutions that can be generated this way. Frequently, modifications introduce rather small changes. A classical example is the two-exchange neighborhood for the traveling salesman problem (TSP), where $N(s)$ consists of all those solutions that can be obtained from s by removing a pair of edges and replacing it by the distinct pair of edges that maintains a tour. Another example for the graph coloring problem is the one-exchange neighborhood of a solution s, which consists of all solutions that can be reached by changing the color of exactly one vertex.

Small modifications have the advantage that their effect is very fast to evaluate and, because of the relatively limited number of neighbors, the neighborhood can be searched rather efficiently. However, the algorithms embedding such neighborhoods have a tendency towards being rather "short-sighted" and they require a large number of search steps to traverse the search space. Therefore sometimes solutions need to undergo some larger, structural changes to improve over the current solution. Such changes are often rather difficult to be managed or found by local search algorithms using small neighborhoods.

Very large-scale neighborhood search techniques try to avoid some of the disadvantages incurred by using small neighborhoods. They allow relatively large modifications of a candidate solution. Therefore, they accept in one search step solutions that are of better quality than those reached within small neighborhoods. Due to the size of very large-scale neighborhoods, often the quality of local minima is much better than that reached in small neighborhoods. Additionally, the ability of making larger modifications to candidate solutions allows for a traversal of the search space in less steps. However, these advantages come at a typically higher computational cost for evaluating the search steps and searching the neighborhood. For many such very large-scale neighborhoods their size increases exponentially with the number of solution components that are allowed to be changed by one single search step. Given this tradeoff between the neighborhood size and the efficiency of searching it, it is difficult to predict the final impact large neighborhoods will have on the performance of more complex stochastic local search algorithms.

The design of algorithms that use very large-scale neighborhoods is based on one of the two main ways of searching improving solutions: exact or heuristic. The exact search is similar to a best-improvement algorithm, i.e. an iterative algorithm that, at each step, tries to move to the best neighboring solution possible. In this case, all neighboring solutions need to be evaluated, at least implicitly. The large size neighborhood can also be examined by some approximate algorithm that does not necessarily identify the best possible

neighbor. An important ingredient for such techniques are heuristic ways of restricting a complete enumeration of the neighborhood. In this latter case, improving moves may be missed.

The next section gives a short review of techniques used to define and search very large-scale neighborhoods (VLSN). We will give concise examples for various such techniques. The largest part of this chapter focuses on the experimental analysis of two examples for the usage of very large-scale neighborhoods. The first example studies algorithms for the *(vertex) graph coloring problem* (GCP); see Sect. 3. Initial experimental results show the desirability of an exact search of the neighborhood with respect to solution quality. However, due to its high computational cost we also study various heuristic schemes for examining the proposed neighborhoods. Finally, we show that despite our efforts, once the very large-scale neighborhood search is integrated into an effective stochastic local search (SLS) algorithm [37] for the GCP, no performance improvements could be observed. The situation is different for the second example, (described in Sect. 4), a very large-scale neighborhood scheme used for tackling the *graph set T-coloring problem* (GSTCP). The GSTCP is an extension of the GCP where sets of colors need to be assigned to each vertex under certain types of color separation constraints. In this case, an exact re-assignment neighborhood, which considers for each vertex a complete color reassignment, was shown to contribute for various GSTCP instance classes towards a new state-of-the-art algorithm. The chapter ends with some concluding remarks and avenues for future research on very large-scale neighborhood search techniques.

2 Searching Large Neighborhoods

Using large size neighborhoods is certainly very appealing. However, these neighborhoods may not be very useful if their complete search requires a very high computational effort. Hence, for making the use of large neighborhoods practical, the neighborhood search problem (NSP), i.e. the problem of finding the best solution in a defined neighborhood set, needs to be efficiently solved. From an abstract perspective, one may distinguish exact algorithms for the NSP and heuristic approaches towards solving the NSP.

As previously mentioned, solving the NSP exactly is akin to best-improvement algorithms. The exact solution of the NSP requires to define neighborhoods that, although exponential in size, admit either efficient, that is, polynomial time algorithms for their exploration, or the exploitation of problem-specific knowledge to speed up the exact search as much as possible. One example of polynomial search is given by dynamic programming approaches like in the *dynasearch* algorithm [18, 19, 49]; we will describe the example application of dynasearch to the TSP in Sect. 2.1. Other examples are network flow based improvement algorithms where the examination of the neighborhood is carried out by solving shortest path or matching problems [30, 34, 56].

Many neighborhood definitions result in NSPs that are themselves $\mathcal{NP}$-hard; this is the case for the cyclic exchange neighborhoods [4, 53, 54] that we discuss in Sect. 2.2. Other examples are local branching for 0–1 integer programming problems where a k-flip neighborhood is defined and searched exactly by solving smaller and more constrained integer programming problems [24], or destruction-reconstruction neighborhoods where the reconstruction is carried out by means of constrained-based tree search [51]. In these cases, the search algorithms might have a significant computational cost, which make it advisable to apply pruning techniques or heuristics to truncate somehow the exact search. This results in a heuristic neighborhood exploration, which does not necessarily identify the best neighboring solution. Hence, there is a trade-off between the scrutiny of the neighborhood search and the solution quality reached.

An important class of very large-scale neighborhood search algorithms are variable depth search (VDS) methods. These work on exponentially sized neighborhoods that are obtained by considering concatenations of simple moves. The number of simple moves that are concatenated to obtain one large move is determined while exploring the neighborhood. VDS methods search the neighborhood in a heuristic way without aiming for the exploration of the full neighborhood. In fact, some of the first very large-scale neighborhood search algorithms were VDS methods. The most famous of these are certainly the Kernighan-Lin heuristic for the graph partitioning problem [41] and the Lin-Kernighan heuristic for the TSP [45]. However, additional VDS algorithms have been proposed rather recently and it should be mentioned that the ejection-chain methods for defining very large-scale neighborhoods [31] share many similarities to VDS. In Sect. 2.3 we will give an overview of the main features of the Lin-Kernighan heuristic for the TSP.

The examples we include below can only give a rough impression of the type of techniques that are available for exploring very large-scale neighborhoods. For a more extensive review of the available techniques, we refer to the overview papers by Ahuja et al. [2, 3].

2.1 Dynasearch

This first example illustrates how an exact algorithm, in this case a dynamic programming algorithm, can be used to search efficiently an appropriately defined, exponentially large neighborhood. In fact, dynamic programming can be used to search neighborhoods of a number of different problems and the resulting class of local search algorithms received the name *dynasearch*. Dynasearch searches neighborhoods that consist of all possible combinations of mutually independent simple search moves. Independence in the context of dynasearch means that the individual simple search moves do not interfere with each other with respect to the objective function and to the constraints of the problem. In particular, the gain incurred by a dynasearch move must be decomposable into the sum of the gains of the simple search moves.

So far, dynasearch has only been applied to problems where solutions can be represented as permutations. Independence between simple search moves is given if the last element involved in one move occurs before the first element of the next move. If $\pi = (\pi(1), \ldots, \pi(n))$ is the current permutation, two moves involving elements from $\pi(i)$ to $\pi(j)$ and from $\pi(k)$ to $\pi(l)$, with $1 \leq i < j \leq n$ and $1 \leq k < l \leq n$ are independent, if either $j < k$ or $l < i$. If, in addition the contributions of the two moves to the solution cost can be computed independently of each other, the best combination of independent moves can be found by a dynamic programming algorithm. (Without loss of generality, we focus here in this chapter on minimization problems.)

Let $\Delta(j)$ be the maximum total cost reduction incurred by independent moves involving only elements from position 1 to j of the current permutation and $\delta(i, j)$ be the cost reduction resulting from a move involving positions between i and j, including i and j. The maximum total cost reduction is obtained either by appending $\pi(j)$ to the current partial permutation or by appending element $\pi(j)$ and applying a move involving element $\pi(j)$ and an element $\pi(i)$ (and possibly elements from $\pi(i)$ onwards).

We assume that the current solution is given by π and set $\Delta(0) = 0$ and $\Delta(1) = 0$. Then, $\Delta(j+1),\ j = 1, \ldots, n-1$ can, in general, be computed in a forward evaluation using the recursive formula

$$\Delta(j+1) = \max \left\{ \max_{1 \leq i \leq j} \{\Delta(i-1) + \delta(i, j+1)\}, \Delta(j) \right\}. \tag{1}$$

The largest reduction in solution cost is then given by $\Delta(n)$ and the single moves to be performed can be found by tracing back the computation steps. In order to apply dynasearch using the forward evaluation, the value of $\Delta(j)$ should not depend on positions $k > j$. If the evaluation of moves depends only on elements $k > j$, a backward version of the dynamic programming algorithm can be applied. This version that starts with $\pi(n)$ and then generates the sequence of $\Delta(j),\ j = n, n-1, \ldots, 1$.

When applying the algorithm, the particularities of the moves have to be taken into account when using Eq. (1). Consider, for example, the application of dynasearch to the TSP using two-exchange moves as the underlying simple moves. Here we assume that a tour is represented as $\pi = (\pi(1), \ldots, \pi(n+1))$, where we define $\pi(n+1) = \pi(1)$. If $1 < i+1 < j \leq n$ and $\pi(j+1) \neq \pi(i)$, a two-exchange move is obtained by removing edges $(\pi(i), \pi(i+1))$ and $(\pi(j), \pi(j+1))$ from the current tour π and introducing edges $(\pi(i), \pi(j))$ and $(\pi(i+1), \pi(j+1))$. The move is accepted if the new tour is shorter, that is, if and only if

$$d(\pi(i), \pi(i+1)) + d(\pi(j), \pi(j+1)) > d(\pi(i), \pi(j)) + d(\pi(i+1), \pi(j+1)),$$

where $d(\cdot, \cdot)$ is the cost function defined on the set of edges.

The dynasearch two-exchange algorithm presented by Congram [18] tries to improve a Hamiltonian path between two fixed end points $\pi(1)$ and $\pi(n+1)$.

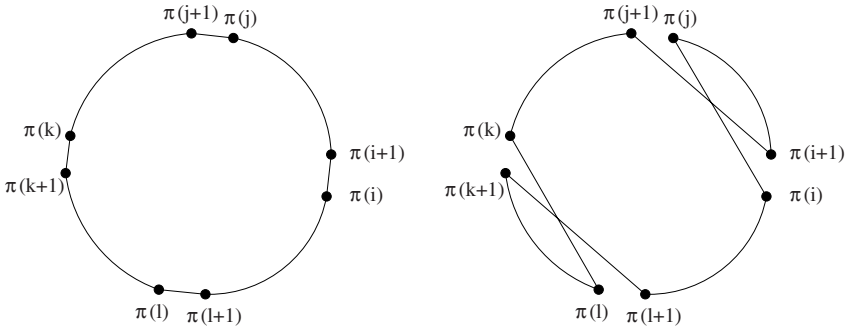

Fig. 1. Example of a dynasearch move that is composed of two independent two-exchange moves

In this case, two two-exchange moves that delete edges $(\pi(i), \pi(i+1))$ and $(\pi(j), \pi(j+1))$, where $1 < i+1 < j \leq n$, $\pi(j+1) \neq \pi(i)$, and $(\pi(k), \pi(k+1))$ and $(\pi(l), \pi(l+1))$, where $1 < k+1 < l \leq n$, $\pi(l+1) \neq \pi(k)$, are independent if we have $j < k$ or $l < i$, because in this case the segments that are rearranged by the two moves do not have any edges in common. An example of such an independent dynasearch move is given in Fig. 1.

In this case, $\Delta(j)$ is the largest reduction of the length of the tour obtained by independent two-exchange moves involving only elements between 1 and j (both included) of the incumbent tour π. The initialization of the dynamic program is $\Delta(1) = \Delta(2) = \Delta(3) = 0$, because we need at least four vertices to apply a two-exchange move. The recursion formula then becomes

$$\Delta(j+1) = \max\left\{ \max_{1 \leq i \leq j-2} \{\Delta(i-1) + \delta(i, j+1)\}, \Delta(j) \right\}$$

and $\Delta(n+1)$ gives the maximal improvement. Dynasearch then repeats these neighborhood searches until no improvement can be found anymore, that is, until a local optimum is reached.

Current applications of dynasearch comprise the traveling salesman problem [18], the single machine total weighted tardiness problem [18, 19], and the linear ordering problem [18]. A general observation from these applications is that dynasearch, on average, is faster than a standard best-improvement descent algorithm and returns slightly better quality solutions. Particularly good performance is reported, if dynasearch is used as a local search routine inside an iterated local search [46]. Currently, iterated dynasearch is the best performing metaheuristic for the single machine total weighted tardiness problem [33] and very good results were also obtained for the traveling salesman problem and the linear ordering problem [18].

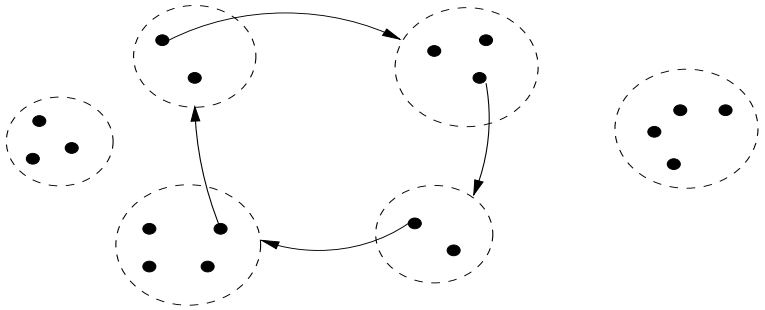

Fig. 2. Example of a cyclic exchange for a partitioning problem

2.2 Cyclic and Path Exchange Neighborhoods

Cyclic and path exchange neighborhoods have been considered as very large-scale neighborhoods for tackling problems where solutions may be represented in some form of a partitioning [4, 5]. Examples of such problems include vehicle routing, capacitated minimum spanning tree, parallel machine scheduling, clustering, and various others.

In a set partitioning problem is given a set W of n elements and a set of subsets $\mathcal{T}$ of W, $\mathcal{T} = \{T_1, \ldots, T_K\}$ such that $W = T_1 \cup \cdots \cup T_K$ and $T_k \cap T_{k'} = \emptyset$, $k, k' = 1, \ldots, K$. A cost $c(T_k)$ is associated with any set T_k. The partitioning problem seeks the partition of W that minimizes the sum of the costs of the subsets in the partition. Formally, the partitioning problem can be defined as:

$$\min c(\mathcal{T}) = \sum_{k=1}^{K} c(T_k)$$
$$\text{s.t. } \mathcal{T} \text{ is a partition of } W.$$

The characteristics of the cost function are not important for the time being; its sole property that needs to be taken into consideration is its separability over subsets.

Frequently, partitioning problems are solved using local search algorithms that are in most cases based on the one- and two-exchange neighborhood. Cyclic exchange neighborhoods are a generalization of the two exchange neighborhood [4, 53, 54] in which instead of swapping only two elements from two subsets, several elements, each belonging to a different subset, are moved (see Fig. 2).

Formally, a cyclic exchange between k subsets (without loss of generality we can assume them to be $T_1, \ldots, T_k$ and the elements we look at to be $a_1, \ldots, a_k$, with $a_i \in T_i$) is represented by a cyclic permutation π of length k, $\pi \neq \mathbf{1}$, where $\pi(i) = j$ means that the element a_i from subset T_i moves into subset T_j. A partition $\mathcal{T}'$ is said to be a *neighbor* of the partition $\mathcal{T}$ if

it is obtained from $\mathcal{T}$ by performing a cyclic exchange and it is feasible with respect to some problem specific constraints. The set of all neighbors of $\mathcal{T}$ defines the *cyclic exchange neighborhood* of $\mathcal{T}$. The cyclic exchange modifies the sets of the partition and therefore their cost. The cost difference for each subset will be the difference between the cost of the subset before performing the cyclic exchange and the cost of the subset after the exchange. The *cost of the cyclic exchange* is the sum of all the cost differences over all subsets in the partition.

In order to find the next move, Ahuja et al. define the *improvement (directed) graph*. The construction of the improvement graph depends on the current feasible partition of the partitioning problem considered. The set of vertices V of the improvement graph is defined as the collection of integers $1, \ldots, n$, each corresponding to an element of the set W. The improvement graph contains the arc (i, j) if the elements corresponding to i and j do not belong to the same subset in $\mathcal{T}$ and the subset to which the element corresponding to j belongs remains feasible after the removal of the element corresponding to j and the addition of the element corresponding to i. We define the partition $\mathcal{U}$ of V to be the collection of subsets $U_1, \ldots, U_K$ corresponding to the subsets in $\mathcal{T}$, that is, the elements of T_k are in one-to-one correspondence with the elements of U_k for each $k = 1, \ldots, K$. Therefore a subset disjoint cycle in the improvement graph, with respect to $\mathcal{U}$, will correspond to a cyclic exchange in $\mathcal{T}$. The arc (i, j) will have an associated cost, equal to the difference between the cost of the set after the removal of the element corresponding to j and the addition of the element corresponding to i, and the cost of the original set that contains the element corresponding to j. Thompson and Orlin [53] showed that there is a one-to-one correspondence between the cyclic exchanges with respect to $\mathcal{T}$ and the subset-disjoint cycles in the improvement graph (with respect to $\mathcal{U}$) and that both have the same cost.

The problem of finding the best neighbor within the cyclic exchange neighborhood can be modeled as a new problem, the *subset disjoint minimum cost cycle problem* (SDMCCP), or in some cases the *subset disjoint negative cost cycle problem* (SDNCCP). The SDMCCP is the problem of finding the minimum cost subset disjoint cycle (a cycle that uses at most one vertex from every subset of the partition of the set of vertices) in the improvement graph. The SDNCCP is the problem of finding a minimum subset disjoint cycle with the additional constraint that its cost is negative. The only real difference between SDMCCP and SDNCCP is that the feasible set of the latter is a subset of the feasible set of the former. Also note that if the network contains negative cycles, then the set of optimal solutions of both problems is the same.

If the SDNCCP has a solution, then this solution corresponds to an improvement in the solution quality of the partitioning problem considered. Hence, the SDNCCP is the problem to be solved when using VLSN in an iterative improvement algorithm. The SDMCCP is important if the VLSN search technique is embedded, for example, into a tabu search procedure;

then finding the best neighbor may be important regardless of whether or not this neighbor is better than the current solution. Ahuja et al. [4] solve the SDNCCP by a heuristic that takes advantage of the fact that only negative cycles are needed. However they make no attempt to solve the SDNCCP exactly. Several exact methods for both SDMCCP and SDNCCP are proposed by Dumitrescu [23]. They can be viewed as generalizations of dynamic programming algorithms commonly used for shortest path problems. The search is accelerated in the case of the SDMCCP by taking advantage of symmetry properties, and in the case of SDNCCP by an elegant lemma by Lin and Kernighan [45], which, as noted also by Ahuja et al. [5], is used to restrict drastically the number of extending paths under consideration. Although the SDNCCP and SDMCCP are $\mathcal{NP}$-hard, the proposed methods are very efficient for the subproblems that arise in practical applications of VLSN search methods. Dumitrescu also proposes some heuristics that are derived from the exact methods by limiting or truncating them in some way.

2.3 Lin-Kernighan Heuristic

While in the previous two examples the underling idea is to search the very large-scale neighborhood exactly, variable-depth search methods are designed with the idea of searching the neighborhood in some heuristic way. The probably best known VDS algorithm is the *Lin-Kernighan (LK) algorithm* for the TSP. It is an iterative improvement algorithm that constructs and systematically tests complex local search moves that are composed of simpler local search steps. The LK algorithm, in particular, is based on complex steps that are generated by a sequence of two-exchange steps.

Let us first explain the mechanism that underlies the construction of the complex search steps. This can be best illustrated by considering the sequence of Hamiltonian paths, i.e., paths which visit each vertex in the given graph G exactly once, which are visited in the move generation. (For an illustration consider Fig. 3.) Initially, a Hamiltonian path between vertices u and v is generated from a tour by removing the edge (u, v). One of the two endpoints, say u, is then kept fixed, while the other will vary. This is the situation depicted in Fig. 3a. In a next step, an edge (v, w) is added, which introduces a cycle into the Hamiltonian path (see Fig. 3b). This resulting structure is also called a δ-path. In the third step, the cycle in the δ-path is broken by removing the only possible edge (w, v') incident to w such that the result is a new Hamiltonian path that could be extended to a tour by adding an edge (v', u) (see Fig. 3c). Instead of closing the tour, another edge can be added that leads to a new δ-path, which is indicated in Fig. 3d. The move construction continues in this way by creating a sequence of δ-paths and tentative intermediate tours.

The heuristic examination of the so generated neighborhood is directed in the LK algorithm by considering the weights of the edges in determining which steps are tentatively tested. The LK algorithms starts from some initial tour s, deletes an edge and determines a δ-path p of minimal weight. If the

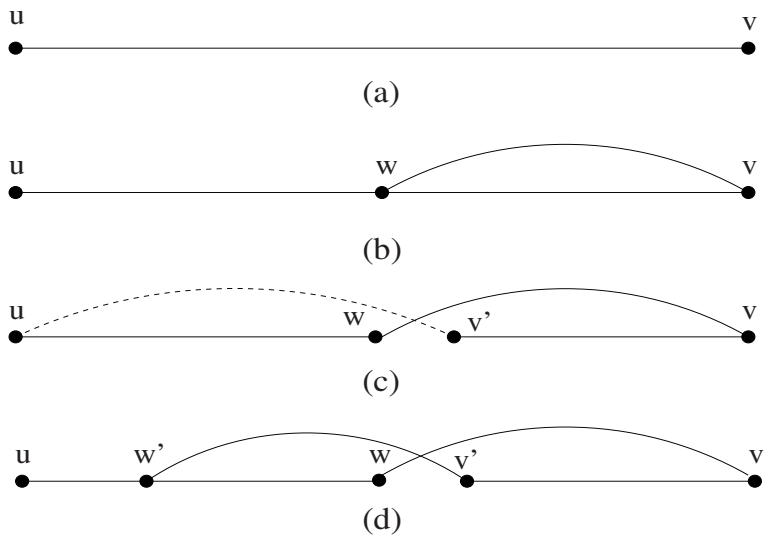

Fig. 3. Schematic view of a Lin-Kernighan exchange step. (a) The original Hamiltonian path; (b) shows a possible δ-path, (c) shows that next Hamiltonian path and (d) shows that a tentative next δ-path

Hamiltonian cycle s' obtained from p in the intermediate steps (as in Fig. 3c) has a weight lower than s, then s' becomes the new incumbent solution. These steps are continued from p and iterated until no δ-path can be obtained with weight smaller than that of the best Hamiltonian cycle found so far in the construction of the complex move. If the best Hamiltonian cycle found in this process improves over the initial solution, it replaces this solution.

In addition to the criterion that stops the construction of complex moves based on the weight of δ-paths and the incumbent solutions, the LK algorithm uses additional tabu criteria. In fact, in the construction of the complex moves any edge that has been added is not removed and no edge that has been removed is added anymore. This rule has the effect that a candidate sequence for a complex step is never longer than the number of vertices in the graph.

In addition to these basic steps, the LK algorithm uses various additional techniques and few exceptions to the above described rule for constructing complex moves. The most important technique is probably a type of backtracking mechanism that allows to consider alternative choices in the edges to be added for constructing the complex moves. Typically, this mechanism is limited to the first few decisions that are done and it is necessary to guarantee that the final tour found by the LK algorithm is locally optimal with respect to the two- and three-exchange neighborhoods. Other mechanisms concern specific types of moves that may be generated are exceptions to the basic construction rules. For a description of these details, we refer to the original

article [45] or to other, more recent descriptions of various LK implementations [7, 39, 48].

VDS algorithms have been used with considerable success for solving a number of problems other than the TSP [41, 47, 57]. Generally, however, the implementation of high-performance VDS algorithms requires considerable effort, especially if large instances are to be tackled.

3 Cyclic and Path Exchange Neighborhoods for Graph Coloring

The graph coloring problem (GCP) is a central problem in graph theory [38] and it arises in a number of application areas such as register allocation [6], air traffic flow management [9], frequency assignment [27], light wavelengths assignment in optical networks [59], and timetabling [21, 43].

In the GCP, one is given an undirected graph $G = (V, E)$, with V being a set of $|V| = n$ vertices and E being a set of edges. A *k-coloring* of G is a mapping $\varphi : V \to \Gamma$, where $\Gamma = \{1, 2, \ldots, k\}$ is a set of $|\Gamma| = k$ integers, each representing one color. A k-coloring is *proper* if for all $(u, v) \in E$ it holds that $\varphi(u) \neq \varphi(v)$; otherwise it is *non-proper*. If for some $(u, v) \in E$ we have $\varphi(u) = \varphi(v)$, the vertices u and v are said to be *in conflict*. (Similarly, we say that edge (u, v) is *in conflict*.) The *conflict set* V^c is the set of all vertices that are in conflict. Alternative to the assignment point of view, a k-coloring can also be seen as a partitioning $\mathcal{C} = \{C_1, \ldots, C_k\}$ of the set of vertices V into k disjoint sets, called *color classes*.

In the optimization version of the GCP, one is asked for the smallest number k of colors such that a proper coloring exists. This number is an intrinsic property of a graph and is called *chromatic number*. The decision version of the GCP asks if, for a given k, a proper coloring exists. This version of the GCP is well known to be $\mathcal{NP}$-complete [40]. It is, hence, not surprising that exact methods for solving the GCP fail to be effective on a significant portion of the known benchmark instances, in part due to their large size [14]. As an alternative, a large number of SLS algorithms have been proposed. Most of these approaches are actually based on a very straightforward one-exchange neighborhood, in which one vertex at a time changes its color class. (See [16] for an overview of local search algorithms for the GCP.) However, the partitioning representation of the GCP directly suggests the application of the cyclic and path-exchange neighborhood that was described in Sect. 2.2. In what follows, we study the performance of SLS algorithms exploiting these neighborhoods and give some insights into their behavior.

3.1 Neighborhoods for the GCP

When tackling the optimization version of the GCP by local search methods, the most used approach is to solve a sequence of k-coloring problems. In a first

step, one guesses an initial value for k, for example, by applying a construction algorithm based on the DSATUR heuristic [12] or the RLF heuristic [43]. Then, each time a proper k-coloring is found, the value of k is decreased by one. This can be done by removing one color and recoloring the vertices without any associated color uniformly at random. If in this iterative process the SLS algorithm cannot find a proper coloring for some k, it returns $k+1$ as an upper bound on the chromatic number.

The evaluation function used by most SLS algorithms counts the number of conflicting edges: $g(\mathcal{C}) = \sum_{i=1}^{k} |E_i|$, where E_i is the set of edges with both end points in C_i. A candidate solution with an evaluation function value of zero corresponds to a proper k-coloring.

In local search, a solution to the GCP is represented by a vector of $|V|$ elements containing the color assigned to each vertex. In addition, for speeding up some of the computations in the local search (for example the neighborhood examination), it is advantageous to maintain a redundant representation consisting of a collection of k sets of vertices corresponding to the k color classes.

The basic *one-exchange* neighborhood operator changes the color of one single vertex. In the partitioning representation, this corresponds to moving one vertex from one color class to another. Clearly, improvements of the evaluation function are only possible if exchanges involve vertices that are in conflict. Hence, in many SLS algorithms the neighborhood is restricted to such vertices and the size of the one-exchange neighborhood is $|V^c| \cdot k$. We denote this neighborhood by N_1. (Note that the size of N_1 changes at run-time of an SLS algorithm with the size of the conflict set.)

Much less frequently used is the neighborhood structure based on the *swap operator*, in which, one vertex in V^c exchanges the color with another vertex in V. Besides being of quadratic size in the worst case (more precisely, of size $|V^C| \cdot |V|$), this neighborhood leaves the cardinality of the color classes unchanged and it is therefore less appealing than N_1.

The partitioning representation of the GCP lends itself to the definition of a *cyclic and path exchange* neighborhood as described in Sect. 2.2. Each partition corresponds to one color class and the cost is the number of conflicting edges in the class. The property of cyclic exchanges to leave the class cardinality unchanged suggests the desirability of allowing also path exchanges.

In the GCP, a *cyclic exchange* of length m acts on a sequence of vertices $(u_1, \ldots, u_m)$ belonging to mutually distinct color classes. For simplicity, we denote the color class of any u_i, $i = 1, \ldots, m$ by C_i and adopt the convention $C_{m+1} = C_1$. The *cyclic exchange* moves then any u_i, $i = 1, \ldots, m$ from C_i into C_{i+1}. The *path exchange*, instead, moves any u_i, $i = 1, \ldots, m-1$ from C_i into C_{i+1} but maintains u_m in C_m. In this latter case, the sequence of exchanges is not closed and the cardinality of C_1 and C_m is modified.

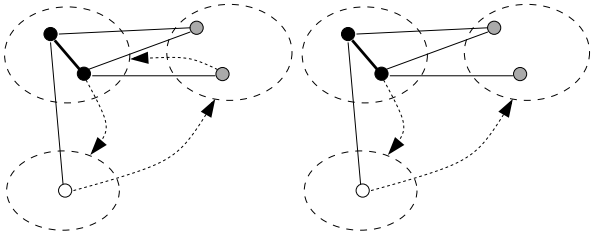

Fig. 4. An example where one-exchange and swap moves do not yield any improvement, while a cyclic (left side) or a path (right side) exchange can be found to yield a proper coloring.

3.2 Cyclic and Path Exchanges and the Number of Local Optima

The new cyclic and path exchange neighborhood for the GCP includes the one-exchange and the swap neighborhood as special cases. Figure 4 gives an example, which shows that by the adoption of the general cyclic and path exchange neighborhoods, improvements over these special cases may be expected. In fact, the one-exchange and swap neighborhoods would fail to yield any neighboring coloring with a better evaluation function value than the coloring depicted in Fig. 4; however, by allowing for cyclic and path exchanges, moves could be found that result in a proper coloring.

Further evidence for the desirability of using the cyclic and path neighborhood is given by experiments on small graphs. In particular, we generated for $n \in \{3, \ldots, 9\}$ all connected, non-isomorphic graphs.[1] In Table 1, we report in the first three columns the number of vertices, the number of such distinct graphs grouped by different chromatic number and for each graph the number of all distinct colorings that use a number of colors between two and the chromatic number.[2] We then applied, starting from these colorings, iterative best improvement algorithms that make use of different neighborhoods (or neighborhood combinations) and for each of these algorithms we counted the number of local optima that do not correspond to proper colorings. For $n \in \{3, \ldots, 7\}$ these experiments were done exhaustively, that is, starting once from each different initial coloring, while for $n \in \{8, 9\}$ we used a random sample of size equal to the number of initial colorings used for the case $n = 7$. The results show clearly that the number of local optima decreases considerably by using a larger neighborhood.

[1] These graphs where generated with the program `geng` of `nauty` by B. McKay, available from `http://cs.anu.edu.au/~bdm/nauty/` (1984-2007, downloaded December 2003).

[2] Given a graph of size n, the number of all colorings that use k colors is given by the Stirling number of second kind $S_{n,k}$ and corresponds to the number of all partitions of n labeled objects (the vertices) into k unlabeled sets (the colors classes). A procedure to generate all these partitions can be found, for example, in [42].

Table 1. Number of distinct local optima that do not correspond to proper colorings for iterative improvement algorithms that differ in the examined neighborhoods. Results pertain all connected, non-isomorphic graphs of different size. Iterative improvement algorithms are run on each graph starting once from each possible distinct coloring that use between two and the chromatic number of colors. For n in $\{8, 9\}$ we considered an overall sample of 421 555 distinct initial colorings.

$\|V\|$	# distinct graphs	# distinct colorings	One-ex	One-ex + swap	One-ex + cyclic	swap + path	cyclic + path
3	2	7	0	0	0	0	0
4	6	61	4	1	1	0	0
5	21	756	25	14	9	0	0
6	112	14113	468	231	148	24	14
7	853	421555	9129	4768	2419	495	265
8	11117	22965511	81944	49416	24879	8006	4913
9	261080	2461096985	133535	85169	42077	18964	11890

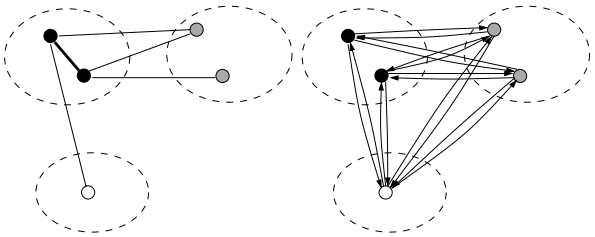

Fig. 5. A graph and a coloring for it (left) and the corresponding improvement graph (right).

3.3 Neighborhood Examination

A crucial element in the examination of the neighborhood is the computation of only local changes in the evaluation of a move. In the one-exchange neighborhood, the evaluation function value of a neighbor $\mathcal{C}'$ of the current coloring $\mathcal{C}$ can be obtained by the value of $\mathcal{C}$: $g(\mathcal{C}') = g(\mathcal{C}) - |A_{C_i}(v)| + |A_{C_j}(v)|$, where $A_{C_i}(v)$ is the set of edges connecting v to other vertices in the class C_i and we assume to move v from C_i to C_j. The value $|A_{C_i}(v)|$ can be obtained in constant time if we record it in an auxiliary matrix $\boldsymbol{\Delta}$ of $|V| \times k$ elements. The initialization of the matrix is done once at the beginning in $\mathcal{O}(|V|^2)$; its update after a one-exchange move is, however, much faster since we need to consider only the entries corresponding to the vertex that moved into a different color class and the vertices adjacent to it. Hence, the worst case complexity for updating $\boldsymbol{\Delta}$ is $\mathcal{O}(|V|)$, although in practice it is much lower.

In the examination of the cyclic and path exchange neighborhood, we can use the improvement graph model and the SDNCCP algorithm devised by Dumitrescu [23]. The improvement graph is the directed graph $G' = (V', D')$, obtained from the graph $G = (V, E)$ and the current color partition $\mathcal{C} =$

$\{C_1, \ldots, C_k\}$. The set of vertices of G' is $V' = \{1, \ldots, n\}$, each vertex in V' corresponding to exactly one vertex $v_i \in V$. The set of arcs D' consists of arcs (i, j) if vertices v_i and v_j belong to two different color classes in $\mathcal{C}$. The set of vertices V' is split into a partition $\mathcal{U}$ of k subsets $U_1, \ldots, U_k$, induced by $\mathcal{C}$, and the elements of U_h are in one-to-one correspondence with the elements of C_h, $\forall h = 1, \ldots, k$ (see Fig. 5). Hence, cyclic and path exchanges in $\mathcal{C}$ will correspond to subset disjoint cycles, respectively paths, with respect to $\mathcal{U}$. The cost associated to an arc (i, j) with $i \in U_i$ and $j \in U_j$ is the cost of inserting vertex i into C_j and removing vertex j from C_j. This cost can be easily computed using the matrix $\boldsymbol{\Delta}$ defined above: $c_{i,j} = |A_{C_j}(v_i)| - |A_{C_j}(v_j)|$. The update of $\boldsymbol{\Delta}$ after each cyclic or path exchange is done by considering the exchanges as a composition of one-exchanges and requires $\mathcal{O}(m|V|))$, where $m \leq k$ is the length of the cyclic or path exchange.

In order to solve the SDNCCP and to find the best cyclic exchange we base the algorithm on the one presented in [23] and another similar one [4]. The algorithm works in a dynamic programming fashion by extending *subset disjoint paths*, that is, paths that visit every subset U_i of the improvement graph $G'(V', D')$ at most once. It is clear that the path can be closed to form a subset disjoint cycle by an arc that connects the last and the first vertex of such a path. This arc always exists given the way the improvement graph is constructed.

We denote a path in the improvement graph by $p = (i_1, \ldots, i_l)$, where i_1 is the start vertex of p, denoted by $s(p)$ and i_l is the end vertex of p, denoted by $e(p)$. We associate a binary vector $w(p) \in \{0, 1\}^k$ with the path p, where $w_j(p) = 1$, if and only if p visits the subset U_j. The cost of p, denoted by $c(p)$, is the total cost of the arcs in the path, that is, $c(p) = \sum_{j=1}^{l-1} c_{i_j, i_{j+1}}$.

We say that the path p_1 *dominates* the path p_2 if $s(p_1) = s(p_2)$, $e(p_1) = e(p_2)$, $c(p_1) \leq c(p_2)$, $w(p_1) \leq w(p_2)$, and the paths do not coincide. This definition will help us to discard paths that are not promising. We define a *treatment* of a path as the extension of that path along all outgoing arcs. After a treatment, a path is marked as *treated*. At any time, only paths that have not previously been treated are selected for treatment. The overall SDNCC algorithm is given in Alg. 16. In the algorithm, q^* contains the ordered sequence of vertices for the most negative cyclic exchange in $\mathcal{C}$ and c^* is its cost.

The algorithm uses a well known lemma from [45] which establishes that if a sequence of edge costs has negative sum, there is a cyclic permutation of these edges such that every partial sum is negative. This allows to restrict the dynamic programming recursion by (i) creating a label of length one only for negative edges (line 2) and (ii) extending a label of length larger than one with a new edge only if the partial sum of the costs associated with the edges remains negative (line 15). The application of this rule does not cause the omission of any promising subset disjoint negative cost cycle. If, on line 11, all the paths in $\mathcal{P}$ are transferred for examination into $\widehat{\mathcal{P}}$ without any restriction, the algorithm is exact and its complexity is $\mathcal{O}(|V'|^2 2^k |D'|)$.

Algorithm 16 to solve the SDNCCP

1: **input:** a graph $G'(V', D')$
2: Let $\mathcal{P}$ all negative cost paths of length 1, i.e., $\mathcal{P} = \{(i,j) : (i,j) \in D', c(i,j) < 0\}$
3: Mark all paths in $\mathcal{P}$ as untreated
4: Initialize the best cycle $q^* = ()$ and $c^* = 0$
5: **for** each $p \in \mathcal{P}$ **do**
6: **if** $(e(p), s(p)) \in D'$ and $c(p) + c(e(p), s(p)) < c^*$ **then**
7: q^* = the cycle obtained by closing p and $c^* = c(q^*)$
8: **end if**
9: **end for**
10: **while** $\mathcal{P} \neq \emptyset$ **do**
11: Let $\widehat{\mathcal{P}} = \mathcal{P}$ be the set of untreated paths
12: $\mathcal{P} = \emptyset$
13: **while** $\exists\, p \in \widehat{\mathcal{P}}$ untreated **do**
14: Select some untreated path $p \in \widehat{\mathcal{P}}$ and mark it as treated
15: **for** each $(e(p), j) \in D'$ s.t. $w_{\varphi(v_j)}(p) = 0$ and $c(p) + c(e(p), j) < 0$ **do**
16: Add the extended path $(s(p), \ldots, e(p), j)$ to $\mathcal{P}$ as untreated
17: **if** $(j, s(p)) \in D'$ and $c(p) + c(e(p), j) + c(j, s(p)) < c^*$ **then**
18: q^* = the cycle obtained by closing the path $(s(p), \ldots, e(p), j)$
19: $c^* = c(q^*)$
20: **end if**
21: **end for**
22: **end while**
23: **for** each $p' \in \mathcal{P}$ subject to $w(p') = w(p)$, $s(p') = s(p)$, $e(p') = e(p)$ **do**
24: Remove from $\mathcal{P}$ the path of higher cost between p and p'
25: **end for**
26: **end while**
27: **return:** a minimal negative cost cycle q^* of cost c^*

Unfortunately the algorithm SDNCC cannot be modified efficiently to search exactly also path exchanges in $\mathcal{C}$. This is due to the way we constructed the improvement graph. Indeed, the cost of a subset disjoint path in the improvement graph does not correspond to the cost of a path exchange in the real graph and an adjustment is required. Given the subset disjoint path $p = (i_1, \ldots, i_l)$ in G', which corresponds to $\tilde{p} = (v_{i_1}, \ldots, v_{i_l})$ in G, the evaluation function of a neighbor $\mathcal{C}'$ obtained from $\mathcal{C}$ after a path exchange given by $\tilde{p}$ is: $g(\mathcal{C}') = g(\mathcal{C}) + c(p) - |A_{C_{i_1}}(v_{i_1})| + |A_{C'_{i_l}}(v_{i_l})|$, where $C'_{i_l} = C_{i_l} \cup \{v_{i_{l-1}}\}$. In order to find the best (most improving) cyclic *or* path exchange in $\mathcal{C}$ we must then modify algorithm SDNCC such that it checks also the cost of paths whenever it checks the cost of cycles, on lines 6 and 17. Moreover, in an exact search we must also avoid to use the lemma of Lin and Kernighan [45], as it would not be anymore valid given that the cost of subset disjoint paths in $\mathcal{U}$

does not correspond to the cost of path exchanges in $\mathcal{C}$. In the final output of the algorithm thus modified, q^* becomes the most negative cyclic or path exchange in $\mathcal{C}$ and c^* its cost.

The algorithm in Alg. 16 returns the most improving neighboring solution for the considered neighborhoods. However, the computational cost of the exact neighborhood exploration may be prohibitively high and therefore various heuristics to prune the search have been implemented. The first is to limit the number of paths that are explored to a maximum of `LAB_LIM` paths. This can be achieved by modifying line 11 of Alg. 16 to let $\widehat{\mathcal{P}} \subseteq \mathcal{P}$ be the set of the `LAB_LIM` cheapest untreated paths. In this way the final solution is not optimal (although it remains very close [23]) but the time complexity drops to $\mathcal{O}(|V'|^2)$. In addition, it is clear that the Lin and Kernighan lemma might be very helpful to further reduce the computational cost of the algorithm. We therefore investigated the usage of the following further rules to prune the search.

Rule 1a. Only subset disjoint negative paths of length one are treated on line 2 and only subset disjoint paths of negative or null cost are treated on line 15.

Rule 1b. Only subset disjoint negative cost paths are treated on both lines 2 and 15 as from lemma of [45].

Rule 2. We introduce the restriction to maintain no more than `LAB_LIM` $< \infty$ paths on line 11.

Rule 3. When considering an iterative improvement procedure, we search for cyclic and path exchanges and hence run the algorithm SDNCC only if the best one-exchange is a non-improving move.

Using the same experimental setting as described Table 1 we compared the number of local optima determined with an exhaustive search of the cyclic and path neighborhood (hence the numbers in the last column of Table 1) against those determined by a search pruned by the combined application of the rules 1a+2+3 and 1b+2+3. In both these two latter cases we could not detect any deterioration of performance, that is, the number of local optima remained the same.

Small graphs, however, are not very helpful to detect the impact of the parameter `LAB_LIM`. Preliminary computational analysis on common benchmark graphs with at least 125 vertices indicated, as expected, that the number of local optima decreases by augmenting `LAB_LIM`. As `LAB_LIM` increases, also the number of iterations to reach local optima decreases; nevertheless, the overall computation time increases strongly. Furthermore, the decrease of the number of local optima becomes less strong as `LAB_LIM` increases. We decided, therefore, to fix `LAB_LIM` to a value of 50 which yields a reasonable trade off between solution quality and running time [14].

In Table 2, we study the impact in computation time for the different neighborhood restrictions in iterative improvement algorithms. The experiments were conducted on uniform random graphs of 125 vertices with three

Table 2. Results on the instances `DSJC125` with edge densities $\delta \in \{0.1, 0.5, 0.9\}$. The table reports for each instance, the performance of iterative best improvement in solving the decision problem associated with different values for k. The indicators for the performance are: the percentage (%) success rate of finding a proper coloring; the median number of conflicting edges $\widetilde{g}$; and the median CPU time expressed in hundredth of seconds to complete a single run (the machine used for the experiment is a 2 GHz AMD Athlon MP Processor with 256 KB cache and 1 GB RAM).

		One-ex			One-ex + swap			cyclic + path			cyclic + path			cyclic + path		
ρ	k							Exhaustive			1a + 2 + 3			1b + 2 + 3		
		%	$\widetilde{g}$	hsec.	%	$\widetilde{g}$	hsec.	%	$\widetilde{g}$	hsec.	%	$\widetilde{g}$	hsec.	%	$\widetilde{g}$	hsec.
0.1	9	74	1	0	93	1	0	100	–	107	100	–	26	100	–	1
	8	29	1	0	64	1	0	100	1	119	100	1	29	100	1	3
	7	2	4	0	12	2	0	88	1	134	91	1	37	87	1	5
	6	0	10	0	0	6	0	12	2	152	14	2	48	12	2	12
	5	0	23	0	0	18	0	0	13	165	0	13	61	0	13	15
0.5	25	1	4	1	7	3	1	100	1	166	82	1	46	96	1	4
	23	0	8	1	0	6	1	96	1	180	53	1	49	81	1	8
	21	0	14	1	0	11	1	56	1	204	16	2	53	33	1	12
	19	0	23	1	0	19	1	0	6	220	0	6	58	0	6	17
	17	0	36	1	0	31	1	0	17	232	0	17	61	0	17	19
0.9	50	0	15	1	0	9	1	63	1	172	3	3	102	22	2	19
	48	0	19	1	0	12	1	35	2	184	1	5	101	10	2	21
	45	0	23	1	0	18	1	0	5	249	0	8	100	0	5	24
	42	0	29	1	0	24	1	0	11	218	0	12	95	0	11	23
	39	0	37	1	0	31	1	0	18	225	0	19	96	0	18	26
	36	0	49	1	0	41	1	0	28	233	0	28	93	0	28	27
	33	0	62	1	0	54	1	0	40	238	0	40	86	0	40	27
	30	0	77	1	0	70	1	0	55	242	0	55	8	0	55	28

different edge densities, namely 0.1, 0.5, 0.9. Results are presented in terms of the ability to solve the various decision problems encountered when decreasing k to the lowest value of k for which a proper coloring is known to exist. At each k, all iterative improvement algorithms are started from the same 1000 initial solutions. The data reported are the success rates for solving the decision problem for each value of k, the median number of conflicting edges in the cases where no proper coloring was found, and the computation time to reach a local optimum. We observe that the contribution of the new neighborhoods remains important also in the large graphs and that it remains pronounced even with a pruned neighborhood examination. Looking at the effect of the two rules 1a and 1b, we observe a considerable difference in computation time and also, rather unexpectedly, higher capacity to solve the problem for the rule 1b, which is therefore to be preferred.

In Table 3 we try to gain deeper insight into which kind of moves are the most important in the new neighborhood. The experimental setting is the same as for Table 1. This time we distinguish two versions for exhaustive

Table 3. Number of moves chosen from each neighborhood. The experimental setting is the same as for Table 1 except for the overall sample of distinct initial coloring that has here size 14 113. In parenthesis is given the maximal length registered for cyclic and path exchanges. Note that a cyclic exchange of length 3 corresponds to $\{v_1, v_2, v_3\}$ and entails that 3 vertices change colors; a path exchange of length 3 corresponds to $\{v_1, v_2, v_3, v_4\}$ and also entails that 3 vertices change colors, as v_4 serves only to determine the arrival color for v_3.

	n	cycle, path exhaustive	path, cycle exhaustive	cycle, path truncated
one-ex	6	24981	24981	24981
	7	29483	29484	29483
	8	34200	34202	34200
	9	38944	38928	38944
swap	6	794	794	794
	7	1138	1138	1138
	8	1489	1491	1489
	9	1859	1861	1859
cyclic	6	370 (3)	65 (3)	370 (3)
	7	743 (4)	103 (3)	743 (4)
	8	1082 (4)	164 (4)	1082 (4)
	9	1449 (5)	218 (4)	1449 (5)
path	6	684 (3)	989 (3)	684 (3)
	7	775 (3)	1415 (3)	775 (3)
	8	936 (3)	1857 (3)	936 (3)
	9	1034 (3)	2276 (4)	1034 (3)

search, depending on whether ties are broken in favor of cyclic or path exchanges (preference is given to the first listed of the two). For cyclic and path exchanges we report also the maximal length of the exchange registered.

By comparing the third and fourth columns in the table, we observe that if paths are taken when they have equal gain as the cyclic exchanges, then the number of cyclic exchanges drops considerably. This indicates that the use of path exchanges alone would miss less gains than the use of cyclic exchanges alone. Hence, the contribution of path exchanges seems higher than the contribution of cyclic exchanges.

The maximal length of cyclic and path exchanges registered indicates that for small graphs of size 9 and number of color classes less or equal to 9, up to 5 vertices may have to change color at the same time in a cyclic exchange and up to 4 in path exchanges. We take this as a further evidence that searching beyond the one-exchange neighborhood might be profitable, as an improving exchange can be found only by moving several vertices. As mentioned, no impact was detected by truncation rules on these small graphs.

3.4 Integrating Cycle and Path Exchanges into SLS Algorithms

The results in Table 2 indicate that the simple iterative improvement algorithms perform rather poorly compared with the results achieved by SLS algorithms. In fact, many SLS algorithms find proper coloring for the smallest values of k indicated in the table within a few seconds on an office PC as of 2005 [14]. It is therefore important to test the use of the new neighborhood within SLS algorithms.

For the GCP, rather simple tabu search algorithms have shown very good and robust performance over a wide range of different graphs. A first tabu search algorithm based on the one-exchange neighborhood was proposed by Hertz and de Werra [36] and it was later improved leading to the best performing tabu search variant [22, 26], which we denote by $\mathsf{TS_{1\text{-}ex}}$. $\mathsf{TS_{1\text{-}ex}}$ chooses at each iteration a best non-tabu or tabu but aspired neighboring candidate solution from the restricted one-exchange neighborhood. If a one-exchange move puts vertex v from color class C_i into C_j, it is forbidden to re-assign vertex v to C_i in the next tt steps; the tabu status of a neighboring solution is overruled if it improves over the best candidate solution found so far (aspiration criterion). If more than one move produces the same effect on the evaluation function, one of those moves is selected uniformly at random. The tabu list length in $\mathsf{TS_{1\text{-}ex}}$ is set to $tt = \mathit{random}(10) + \delta \cdot |V^C|$, where $\mathit{random}(10)$ is an integer uniformly chosen from $\{0, \ldots, 10\}$ and δ is a parameter typically set to 0.5. Since tt depends on $|V^C|$, the tabu list length varies dynamically with the evaluation function value.

To test the usefulness of the cyclic and path exchange neighborhoods, we integrated them into a tabu search algorithm, which we denote as $\mathsf{TS_{VLSN}}$. $\mathsf{TS_{VLSN}}$ is closely related to $\mathsf{TS_{1\text{-}ex}}$. It first selects the best non-tabu move in the one-exchange neighborhood. If this move has the same evaluation function value as the current one, the cyclic and path exchange neighborhood is searched. In all other cases, that is, if the best non-tabu one-exchange leads to an improvement or a deterioration of the evaluation function, the one-exchange move is applied and the tabu list is updated.

The algorithm SDNCC is modified by including in the initial set $\mathcal{P}$ of line 2 only paths of length one consisting of at least one vertex from V^c. The tabu mechanism is applied to the search for cyclic and path exchanges by discarding any neighboring candidate solution that involves the reassignment of a vertex to some color class that is currently tabu. The tabu list is updated by considering the path or the cyclic exchange as a composition of one-exchanges and the value of tt for a specific vertex–color class pair (v, i) is chosen using the rule of $\mathsf{TS_{1\text{-}ex}}$. Yet, contrarily to $\mathsf{TS_{1\text{-}ex}}$, $\mathsf{TS_{VLSN}}$ does not use an aspiration criterion. This allows to discard vertices that are tabu very soon in the search for subset disjoint paths.

The comparison of SLS algorithms is generally performed on the set of benchmark instances available at `http://mat.tepper.cmu.edu/COLOR04`. This repository includes quite heterogeneous graphs and a full account of the

results is given in [14]. Here, we focus on two types of random graphs: geometric (G) graphs and uniform (U) graphs. The geometric graphs are generated from points in a two dimensional grid with random, integer coordinates in $[0, 1)$. There is a one-to-one correspondence between points and vertices in the graph. Edges are assigned between pairs of vertices if the Euclidean distance between the corresponding points is less or equal to a value d. The uniform graphs are generated by including each of the $\binom{|V|}{2}$ possible edges independently with probability p. For both types of graphs, we fixed the parameter d and p such that three different edge densities are considered: 0.1, 0.5 and 0.9. A number of graphs (see strip text in the plots of Fig. 6) with 1000 vertices are generated and the algorithms are run once on each instance with a fixed time limit, equal for all algorithms. The time limit of 150, 700, and 1500 seconds[3] for graphs of density 0.1, 0.5, 0.9, respectively, was chosen so as to allow $\mathsf{TS}_{\mathsf{1\text{-}ex}}$ to accomplish about $10000 \times |V|$ iterations. Further improvements after such long runs become unlikely for $\mathsf{TS}_{\mathsf{1\text{-}ex}}$. As reference, we add in the analysis two algorithms that have shown the best results for the classes of instances studied. These are the hybrid evolutionary algorithm [26] and a guided local search [14].

In Fig. 6, we report the results of a rank-based analysis. In particular, we rank within each instance the results in terms of the lowest k for which a proper coloring is found by each algorithm. In this way, the problem of different instance scales is removed. We then aggregate the ranks over the instances by computing the average rank for each algorithm. A lower average rank means that the algorithm performs better. Moreover, we use simultaneous confidence intervals extended around the average values to provide an indication of the statistical significance of the differences. These intervals are determined by an all-pairwise comparison procedure carried out through the Friedman statistical test at a 5 % protected level of significance [15],[20, p. 371]. In the figure, two algorithms are significantly different if the confidence intervals of the corresponding average ranks do not overlap.

These results indicate that the usage of the cyclic and path exchange neighborhoods does not result into improved performance of the tabu search algorithm. On the contrary, for the uniform graphs the performance of $\mathsf{TS}_{\mathsf{VLSN}}$ is significantly worse than that of $\mathsf{TS}_{\mathsf{1\text{-}ex}}$. The same is true for the geometric graphs of density 0.5. Only on the geometric graphs with densities 0.1 and 0.9, $\mathsf{TS}_{\mathsf{VLSN}}$ yields statistically similar performance to $\mathsf{TS}_{\mathsf{1\text{-}ex}}$. Hence, we obtain the opposite result to what we may have expected from the previous analysis.

A possible explanation for this result is given in Fig. 7, where we depict the behavior of $\mathsf{TS}_{\mathsf{1\text{-}ex}}$ and $\mathsf{TS}_{\mathsf{VLSN}}$ from two different perspectives: over time and over iteration number. It can be observed that $\mathsf{TS}_{\mathsf{VLSN}}$ remains competitive comparing results based on an iteration number, but it is not competitive when comparing the development of the solution quality based on computation

[3] Times refer to a computer with 2 GHz AMD Athlon MP 2400+ Processor with 256 KB cache and 1 GB of RAM memory.

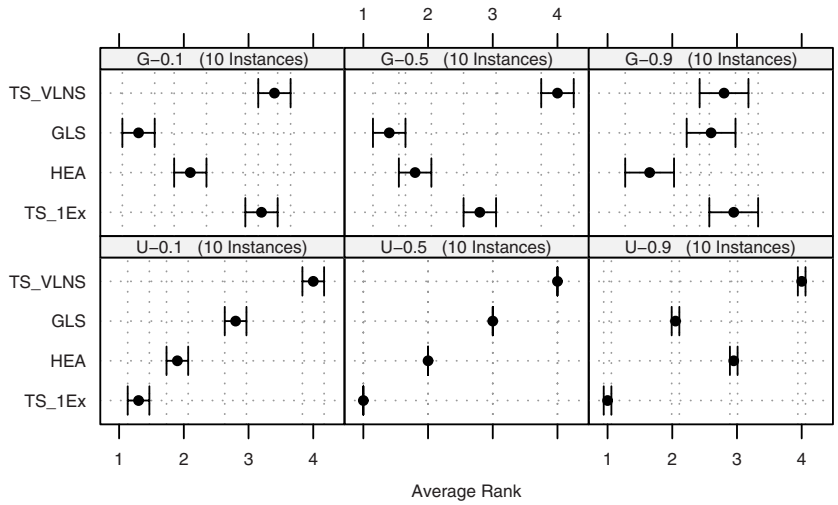

Fig. 6. The 95 % confidence intervals for the all pairwise comparisons of SLS algorithms on aggregated geometric (indicated by G) and uniform (U) random graphs in the GCP. The x-axis indicates the average rank while the confidence intervals are derived from the Friedman test. The more the interval is shifted towards the left the better is the algorithm performance.

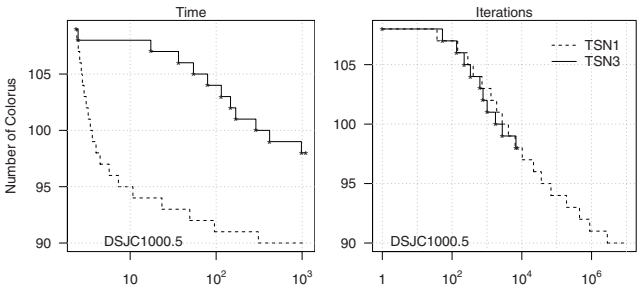

Fig. 7. The development of the solution quality of $\mathsf{TS}_{\text{1-ex}}$ and $\mathsf{TS}_{\text{VLSN}}$ over time (left) and number of iterations (right). Given is the development of the median number of colors across 10 runs per algorithm.

time. This is due to the high added computational cost of searching the cyclic and path exchange neighborhoods. Ideally, this increased computation cost of the neighborhood exploration should be payed off by a faster improvement in solution quality. Although this seems to be the case in the right figure, where the curve of solution cost for $\mathsf{TS}_{\text{VLSN}}$ is slightly lower than the one for $\mathsf{TS}_{\text{1-ex}}$, this minor improvement is probably not enough to yield any concrete advantage inside $\mathsf{TS}_{\text{1-ex}}$ for the GCP.

A cause for the weak performances of $\mathsf{TS}_{\mathsf{VLSN}}$ might be the inefficient use of the SDNCC algorithm in the search for path exchanges in $\mathcal{C}$. In a late discovery we found a way to improve on this issue. In detail, the construction of the improvement graph can be modified by adding k vertices, $v_{n+1}, \ldots, v_{n+k}$ one for each of the U_i partitions. These are "dummy" vertices in the sense that they do not correspond to any vertex in V but are useful to represent the cases in which a vertex enters in a class C_i and no vertex leaves it. The cost of the arcs connecting the new vertices with other vertices of G' not in the same class is defined by: $c_{i,n+l} = |A_{C_l}(v_i)|$ for $l = 1, \ldots, k$; $c_{n+l,j} = -|A_{C_j}(v_j)|$ for $l = n+1, \ldots, n+k$; and $c_{n+h,n+l} = 0$ for $l, h = 1, \ldots, k$. In this way it is possible to model in the improvement graph the costs of path exchanges and the lemma by Lin Kernighan in the SDNCC algorithm turns valid again. Note that the inclusion of edges $(n+h, n+l)$ for $l, h = 1, \ldots, k$ with cost zero is not necessary but it permits to find, eventually, more than one independent exchanges in a single run of the SDNCC algorithm (in a dynasearch fashion). However, in spite of all these favorable premises, preliminary experiments resulted in the same conclusions, that is, $\mathsf{TS}_{\mathsf{VLSN}}$ does not improve over $\mathsf{TS}_{\mathsf{1\text{-}ex}}$.

3.5 Other Studies in the Literature

Beside the work presented here, there are few other studies in the literature on very large-scale neighborhood structures for the GCP.

Glass and Prügel-Bennet [29] observed that permuting the colors within a subgraph does not affect the contribution to the evaluation function of the subgraph itself. They devised a permutation neighborhood that first determines a subgraph that defines a partition of the graph and a cut of the edges joining vertices in the subgraph to vertices outside the subgraph. Then, the permutation of colors within the subgraph that minimizes the number of conflicting edges in the cut is found by solving a matching problem in a suitably defined bipartite graph. Tests on this neighborhood structure used together with the one-exchange and enhanced by a perturbation mechanism in an iterated local search fashion failed to show this approach to be competitive.

Gonzalez-Velarde and Laguna [32] proposed an ejection-chain neighborhood which implements cyclic exchanges of vertices but they limit the chains to maximum length 3. Avanthay, Hertz and Zufferey [8] propose several distinct neighborhoods some of which are of exponential size. Nevertheless they do not provide any insight on how to search them efficiently. In none of these cases results are competitive.

Trick and Yildiz [55] devise α-β-swaps and α-expansions. The former consist of exchanging the color of a subset of vertices that belong to two different color classes defined by α and β. The latter consists of changing the color of any set of vertices to α. It is shown that finding the best neighbor in both these neighborhoods corresponds to solving a MAX-CUT problem in an opportunely defined graph. In the experiments the authors use a heuristic

algorithm for solving the MAX-CUT problem. Although promising, the preliminary results remain inferior to the state-of-the-art.

Other neighborhoods, though not of exponential size, have been studied in Gendron, Hertz, and St-Louis [28]. They build on the fact that the problem of finding the chromatic number of a graph is equivalent to the problem of finding a circuit-free orientation of the edges of a graph so that the length of the shortest path in the resulting digraph is minimum. Hence they study four neighborhoods in the edge orienting problem and derive graph theoretical properties that allow a polynomial search of each neighborhood. In the results presented, the best neighborhood consists of the reversal of a single arc on a longest path of the current orientation. However, again, the results are not competitive with the state-of-the-art.

We can therefore say that various types of very large-scale neighborhoods have been studied for the GCP but none of these studies could clearly identify any positive impact of these very large-scale neighborhoods for improving state-of-the-art algorithms for the GCP. Hence, our study adds further evidence that the improvement in the solution quality per local search step given by the usage of very large-scale neighborhoods does not pay off the additional computation time required for searching the neighborhoods for the GCP.

4 Color Reassignment Neighborhood for Graph Set T-Coloring

The *graph set T-coloring problem* (GSTCP) generalizes the *graph coloring problem* in the sense that it asks for the assignment of sets of integers to the vertices of a graph. The assignment must satisfy constraints on the separation of any two numbers assigned to a single vertex or to adjacent vertices. The GSTCP arises in the modeling of various real-life problems, the most important being the assignment of frequencies to radio transmitters when designing mobile phone networks. In this case, vertices represent transmitters and colors the frequencies to be assigned to the transmitters subject to certain interference constraints, the T-constraints [35]. Other applications stem from traffic phasing and fleet maintenance [50], or task assignment in which a large task is partitioned into incompatible subtasks and a set of time periods needs to be assigned to each subtask so that incompatible subtasks are in different time periods [52].

More formally, in the GSTCP we are given: (i) an undirected graph $G = (V, E)$, where V is the set of $n = |V|$ vertices and E is the set of edges, (ii) a set Γ of nonnegative integer numbers (called colors), (iii) a number $r(v)$ of required colors at each vertex $v \in V$, and (iv) a collection T (called T-set) of nonnegative integers including zero, such that there is an integer t_{uv} for each edge $(u, v) \in E$ and an integer t_u for each vertex $u \in V$ representing the allowed *separation distances* of colors between and within vertices. The

decision version of the GSTCP asks for a mapping $\varphi : V \to \mathcal{P}(\Gamma)$ such that the three groups of constraints

$$|\varphi(v)| = r(v) \qquad \forall\, v \in V \tag{2}$$

$$|x - y| \geq t_u \qquad \forall\, u \in V,\ \forall\, x, y \in \varphi(u), x \neq y \tag{3}$$

$$|x - y| \geq t_{uv} \qquad \forall\, uv \in E,\ \forall\, x \in \varphi(v),\ \forall\, y \in \varphi(u) \tag{4}$$

are satisfied. We call such a multi-valued function φ a *proper set T-coloring*, if all these constraints are satisfied and *improper*, otherwise. The three groups of constraints to be satisfied are called *requirement constraints*, *vertex constraints*, and *edge constraints*, respectively.

We consider the optimization version of the GSTCP in which we are asked for a proper set T-coloring of minimal span, that is, a φ of minimal maximal difference between the colors used, $\min_\varphi \max_{u,v \in V}\{|x - y| : x \in \varphi(u), y \in \varphi(v)\}$. Without loss of generality we fix $\min_{u \in V}\{x : x \in \varphi(u)\}$ to 1. Hence, $\Gamma = \{1, 2, \ldots, k\}$ and the span is $k - 1$. Our task is minimizing k.

4.1 Neighborhood Definitions

A possible approach for applying local search to the GSTCP is to solve a sequence of decision problems, in which a proper set T-coloring is searched for a fixed number k of available colors. This approach has been shown in the literature to be preferable to some others [14]. In this case, a solution is represented by a vector of colors of length $\sum_{v \in V} r(v)$ and an array of pointers indicating where the set of colors assigned to each vertex starts. The effective search space is reduced to only those candidate colorings that satisfy the vertex constraints [14]. Hence, requirement constraints and vertex constraints are always satisfied and the evaluation function needs only to count the number of unsatisfied edge constraints.

The standard one-exchange neighborhood N_E is defined by the operator that changes a single color at a vertex without breaking any vertex constraint. We also studied a new very large-scale neighborhood for the GSTCP. This neighborhood is defined by the operator that reassigns all the colors of one single vertex. In particular, the reassignments are those that satisfy the requirement, vertex and edge constraints acting on the vertex. As such, the application of this operator can be seen as an exact solution to the subproblem of finding an assignment of k colors to one vertex such that no constraint acting on this vertex is violated and no other vertex changes its color assignment. Clearly, given a current configuration, it is possible that for a vertex a reassignment of colors that satisfies all constraints on that vertex does not exist. In this case, if only vertices involved in at least one conflict are considered, the neighborhood is empty. We call this neighborhood the vertex color reassignment neighborhood and we denote it by N_R.

4.2 Neighborhood Examination

Next, we describe the algorithm for the examination of N_R. First a vertex, involved in at least a conflict, is chosen. Then a set F is constructed, which contains the colors that are feasible with respect to the constraints acting on the vertex. This can be done in $\mathcal{O}(|V|k)$ if standard speed-up techniques from the GCP are used [25]. Finding $r(v)$ colors in F that satisfy the vertex constraints, that is finding one neighbor in N_R, is easy: order the values in F, scan F, and remove all colors that are not distant enough from the previous ones. This procedure is deterministic and, unfortunately, it yields the same color reassignment if a vertex is visited twice and its adjacent vertices have not changed the colors. To avoid this cycling behavior, one may introduce some randomization in the reassignment and, hence, when visiting a vertex a second time, a different configuration is obtained, which possibly can be profitably propagated. An implementation of this strategy requires to determine all subsets of F of size $r(v)$ that satisfy the vertex constraints and to pick one at random. More formally, this problem can be formulated as finding a *subsequence of length L of H integers with mutual distance not smaller than D:* given an arbitrary sequence of integers $s = \{s_1, \ldots, s_H\}$, find a subsequence $l = \{l_1 \ldots l_L\}$ of length L with $l_i \in s$, $\forall\, i = 1, \ldots, L$ and such that the mutual distances are not smaller than D, that is, having $|l_i - l_j| \geq D$, $\forall i, j$. Hence, finding all such subsets of F is the same problem as *enumerating all subsequences of length L of H integers with mutual distance not smaller than D.*

It is possible to use solve this problem using a dynamic programming style algorithm, which solves subproblems and stores their solutions in a table. If we have an ordered sequence of integers in F, a proper coloring can be seen as an ordered subsequence of integers that is itself composed of other subsequences. Each of these subsequences, in turn, can be extended, in principle, in a number of different ways to a sequence of length $r(v)$ by adding elements from F. The total number of possible extensions for each subsequence can be defined recursively. Once this is done, it is possible to choose one solution randomly.

More formally, let s be an ordered vector of integers in F, $L = |F|$, $D = t_v$ and $H = r(v)$. For each position i of s we have $next(i) = \min_j\{j : j > i,\ s[j] - s[i] \geq D\}$. For each subsequence l of s, the number $M_H[i]$ of proper subsequences of s of length $h \in \{1, \ldots, H\}$ containing $l = \{s[1], \ldots, s[i]\}$, can be computed recursively as

$$M_h[i] = \begin{cases} L - i + 1, & \text{if } h = 1 \\ M_{h-1}[next(i)] + M_h[i+1], & \text{if } L - i - 1 > h \geq 2 \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

if $i \geq 1$ and $M_h[0] = 0$ by convention.

The total number of proper subsequences of length $r(v)$ is given by $M_H[1]$. For selecting one at random, one may scan the sequence s and select each element with probability $M_{h-1}[next(i)]/M_h[i]$, that is, the fraction of extensions

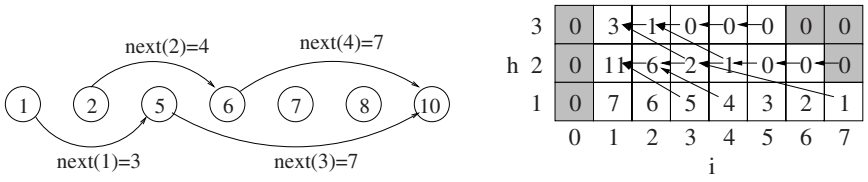

Fig. 8. Given is an example for a vertex exact color reassignment where we have that $F = \{1, 2, 5, 6, 7, 8, 10\}$, $L = |F| = 7$, $D = t_v = 4$ and $H = r(v) = 3$. On the left is given the vector s of 7 integers. For each integer in the sequence the pointer $next()$ is computed; where it is not otherwise indicated, it is set to 0. On the right is given the table of $M_h[i]$ values. This table is filled starting from the low right corner; arrows indicate which already stored values are used to determine new table entries. The grey cells indicate cells that are not used in the computation and, hence, are assigned values by convention.

that contain the element in question. If an element is chosen, the scan continues from $next(i)$. Clearly, the sequence of numbers can be generated in linear time using the information provided by $M_h[i]$. However, computing the recursion (5) for all h and i takes exponential time. Fortunately, this can be done more efficiently by iteratively computing the values $M_h[i]$ at the beginning and recording them in a table. This iterative process is illustrated in Fig. 8, right: the table is completed going from bottom to top and from right to left within each row. Each new table entry requires the values of $M_{h-1}[next(i)]$ and $M_h[i+1]$, which are already computed and stored. Hence, the $next$ function and the table can be computed in $\mathcal{O}(\max(D, H, \log L) \cdot L)$.

4.3 Experimental Results with SLS Algorithms

To assess the contribution of the new neighborhood, we consider its use within more complex SLS algorithms. As for the GCP, we based the new algorithm on a tabu search algorithm that uses in addition a standard one-exchange neighborhood (N_1). We compare the final algorithms with our reimplementation of the squeaky wheel algorithm that was shown to be effective for this problem [44].

With N_1 we use a standard Tabu Search procedure that chooses at each iteration a best non-tabu move or a tabu but "aspired" neighboring solution from the one-exchange neighborhood. The tabu list forbids to reverse a move and the tabu tenure is chosen as $tt = random(10) + 2\delta|V^c|$, where V^c is the set of vertices which are involved in at least one conflict, δ is a parameter, and $random(10)$ is an integer random number uniformly distributed in $[0, 10]$; this choice follows that of a successful tabu search algorithm for the graph coloring problem [26] and yields an algorithm analogous to that proposed in [22]. We denote this algorithm $\mathsf{TS}_{\text{1-ex}}$.

When applying Tabu Search with N_R, a heuristic rule is used to reduce the effort of neighborhood exploration. At each iteration, first the best non-tabu

move in N_1 is determined. If it improves on the current solution, it is accepted. If it leaves the evaluation function value unchanged or worsens it, a move is searched in N_R, restricted to vertices involved in at least one conflict. If a proper reassignment is found, it is applied; otherwise the best non-tabu move in N_1 is applied. In fact, the tabu search mechanism is applied only to moves in N_1 and it is in all equal to $\mathsf{TS}_{\mathsf{1\text{-}ex}}$. The randomized examination of N_R implements already an anti-cycling mechanism and this is the reason why preliminary experiments indicated the use of randomization preferable to that of determinism. We call the overall algorithm TS+R.

We also developed a variant of TS+R that considers a color reassignment for a randomly chosen vertex from V if no move is found in N_R restricted to conflicting vertices. This is motivated by the fact that a random reassignment of colors to vertices where no conflict is present may produce a change that can propagate profitably. We denote this variant TS+R*.

There are four sets of instances for benchmarks on this problem. We refer to [17] for a complete list of references and results on all classes of instances. Here, we restrict ourselves to presenting results on random uniform graphs of size $n = 60$. In such graphs, each of the $\binom{n}{2}$ possible edges is present with a probability $p = \{0.1, 0.5, 0.9\}$. Vertex requirements are chosen uniformly from the set $\{1, \ldots, r\}$ and vertex and edge separation distances are chosen uniformly from the set $\{1, \ldots, t\}$.

First we examine whether the reassignment neighborhood impacts the performance of tabu search. For this, we give in Table 4 the number of improvements due to a move in N_R in comparison to improvements in N_1. We consider 10 instances for different classes of uniform graphs and report the median values derived from one run of TS+R on each single instance. The algorithm was stopped after 10 000 iterations. The first column reports the number of iterations in which an improving solution was found in N_1 while the second column reports the number of improving solutions that was found in N_R. Note that this latter case occurs only if no possible improvement was found in N_1 while an improvement was still possible in N_R. Hence, the second column indicates a positive contribution in terms of number of improvements of the new neighborhood. The third column gives an indication of how many times the neighborhood N_R was searched in 10 000 iterations. We count a search for each attempted reassignment of colors to a single vertex. Finally, the last two columns report the values of $|F|$ and the span of integers in F in order to give an indication of the size of problems being solved.

We then compare the five SLS algorithms on several classes of instances. In [14], it was shown that the primary source of differences in the relative performance of the algorithms under study is the edge density of the graphs, while the combinations of values for r and t resulted to be less important (the study was limited however to values of r and t smaller or equal 10, and further investigation would be needed for larger values). Here, we fix $r = 10$ and $t = 5$ and consider graphs with edge density of $\{0.1, 0.5, 0.9\}$. In our experiments, we impose the same computation time limit for all algorithms. This is necessary

Table 4. Median statistics from 10 000 iterations of TS+R on 10 instances per class. The first two columns report the improvements found in N_1 and N_R, respectively. The third column reports the number of single-vertex searches in N_R. The last two columns report the number of colors in F, that is, the colors which are feasible according to the edge-distance constraints of vertices adjacent to v, and the range of colors in F, that is, $f_M - f_m$, with $f_M = \max\{c : c \in F\}$ and $f_m = \min\{c : c \in F\}$.

class	# improv. in N_1	# improv. in N_R	single-vertex searches in N_R	$\lvert F\rvert$	$f_M - f_m$
size=60, dens.=0.1, $r = 5, t = 5$	1179	59	17698	7	25
size=60, dens.=0.1, $r = 5, t = 10$	839	98	15953	8	50
size=60, dens.=0.1, $r = 10, t = 5$	6	0	20110	26	73
size=60, dens.=0.5, $r = 5, t = 5$	776	194	25463	6	43
size=60, dens.=0.5, $r = 5, t = 10$	926	113	22112	6	133
size=60, dens.=0.5, $r = 10, t = 5$	851	138	35321	9	69
size=60, dens.=0.9, $r = 5, t = 5$	651	117	21370	5	53
size=60, dens.=0.9, $r = 5, t = 10$	223	30	13782	8	159
size=60, dens.=0.9, $r = 10, t = 5$	640	77	23402	10	209

because the single iterations of $\mathsf{TS}_{\text{1-ex}}$ and TS+R have different computation time requirements. The time limit was chosen such that $\mathsf{TS}_{\text{1-ex}}$ could accomplish $I_{max} = 10^5 \times \sum_{v \in V} r(v)$ iterations. More details on the time limit are reported in [14]. Before running the experiments, each algorithm was fine-tuned using the F-race algorithm [10, 11], which is a fully automatic tuning procedure based on sequential testing. In the tabu search algorithms the only parameter to be determined is δ. For each of the algorithms we used numbers in $\{0.5; 1; 10; 20; 30; 40; 50; 60; 70; 100\}$ as candidates; the best value found was 10 for the random uniform instances. In what follows, each algorithm uses this value for δ.

We carried out the experiments and the analysis in a similar way as discussed in Sect. 3.4. That is, we run each algorithm once on each instance and ranked within the instance the result expressed in terms of color span. In Fig. 9 we report the analysis through simultaneous confidence intervals derived from the Friedman test [15, 20] at a 5 % protected level of significance. The more the interval is shifted towards the left the better the performance of the algorithm is. We made a different analysis for each of the three instance classes determined by edge density.

Results vary among the three classes of instances. On the low density instances the vertex color reassignment neighborhood does not introduce any significant improvement and, in fact, it may even worsen the basic $\mathsf{TS}_{\text{1-ex}}$ slightly. However, as the edge density increases, using this new neighborhood becomes worthwhile and TS+R results to be a new state-of-the-art algorithm. Hence, TS+R is an example of an SLS algorithm where a very large-scale neighborhood gives a significant contribution towards enhancing performance.

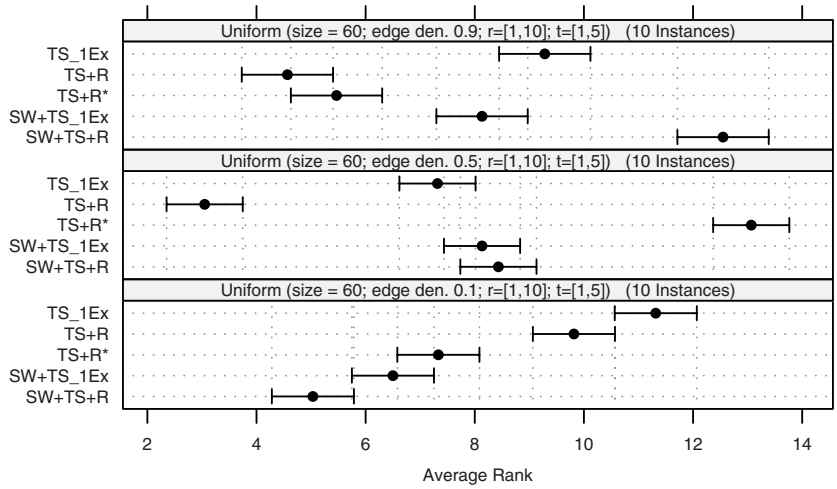

Fig. 9. The 95 % confidence intervals for the all-pairwise comparisons of SLS algorithms on aggregated uniform random instances of the GSTCP. The x-axis indicates the average rank while the confidence intervals are derived from the Friedman test

5 Conclusions

Very large-scale neighborhood searches have received a significant attention, especially in the last few years. Many different ways of defining and searching neighborhoods have been proposed and for various problems, very large-scale neighborhood searches are a central part of state-of-the-art SLS algorithms. Probably the best known example is the Lin-Kernighan heuristic for the traveling salesman problem. For several other problems, including generalized assignment [58], scheduling problems [33], or capacitated minimum spanning trees [5] excellent results have been reported. However, the design and implementation of very large-scale neighborhood searches requires significant insights into the particular problem being tackled, the usage of efficient data structures, and often substantial implementation time. In addition, despite significant efforts, for various problems rather negative results have been obtained in the sense that very large-scale neighborhood search algorithms failed to improve algorithmic performance. One such example is the graph coloring problem, which we also treated in this chapter, or the specific scheduling problem reported in [13]. Certainly, the list of negative examples may be longer – but unfortunately, often such negative results are not published or made available in some other form to the research community.

Given this overall picture, it seems that no clear prediction can be made as to when a very large-scale neighborhood search would result into a highly performing algorithm. This is in part due to missing guidelines as to how

to use very large-scale neighborhood searches and the development of such algorithms is much left to the intuition of the researcher. Possibly, by some preliminary examination of the trade-off between computation time and solution quality improvement incurred by a very large-scale neighborhood some more directed choice may be done. Hence, the usage of very large-scale neighborhoods appears to be a promising possibility but there are still a large number of open research questions to be investigated.

Acknowledgments

Thomas Stützle acknowledges support from the Belgian F.R.S.–FNRS of which he is a research associate.

References

1. E. H. L. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Chichester, UK, 1997.
2. R. K. Ahuja, O. Ergun, J. B. Orlin, and A. P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1–3): 75–102, 2002.
3. R. K. Ahuja, Ö. Ergun, J. B. Orlin, and A. P. Punnen. Very large-scale neighborhood search: Theory, algorithms, and applications. In T. F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics*, pages 20-1–20-15. Chapman & Hall/CRC, Boca Raton, FL, USA, 2007.
4. R. K. Ahuja, J. B. Orlin, and D. Sharma. Very large-scale neighbourhood search. *International Transactions in Operational Research*, 7(4–5):301–317, 2000.
5. R. K. Ahuja, J. B. Orlin, and D. Sharma. Multi-exchange neighborhood search algorithms for the capacitated minimum spanning tree problem. *Mathematical Programming*, 91(1):71–97, 2001. Series A.
6. M. Allen, G. Kumaran, and T. Liu. A combined algorithm for graph-coloring in register allocation. In D. S. Johnson, A. Mehrotra, and M. Trick, editors, *Proceedings of the Computational Symposium on Graph Coloring and its Generalizations*, pages 100–111, Ithaca, New York, USA, 2002.
7. D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Finding tours in the TSP. Technical Report 99885, Forschungsinstitut für Diskrete Mathematik, University of Bonn, Germany, 1999.
8. C. Avanthay, A. Hertz, and N. Zufferey. A variable neighborhood search for graph coloring. *European Journal of Operational Research*, 151(2):379–388, 2003.
9. N. Barnier and P. Brisset. Graph coloring for air traffic flow management. *Annals of Operation Research*, 130:163–178, 2004.
10. M. Birattari. The race package for R. Racing methods for the selection of the best. Technical Report TR/IRIDIA/2003-37, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2003.
11. M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In W. B. Langdon et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, pages 11–18. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2002.

12. D. Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.
13. T. Brueggemann and J. L. Hurink. Two very large-scale neighborhoods for single machine scheduling. *OR Spektrum*, 29:513–533, 2007.
14. M. Chiarandini. *Stochastic Local Search Methods for Highly Constrained Combinatorial Optimisation Problems*. PhD thesis, Computer Science Department, Darmstadt University of Technology, Darmstadt, Germany, August 2005.
15. M. Chiarandini, D. Basso, and T. Stützle. Statistical methods for the comparison of stochastic optimizers. In K. F. Doerner, M. Gendreau, P. Greistorfer, W. J. Gutjahr, R. F. Hartl, and M. Reimann, editors, *MIC2005: The Sixth Metaheuristics International Conference*, pages 189–196, Vienna, Austria, August 2005.
16. M. Chiarandini, I. Dumitrescu, and T. Stützle. Stochastic local search algorithms for the graph coloring problem. In T. F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics*, pages 63-1–63-17. Chapman & Hall/CRC, Boca Raton, FL, USA, 2007.
17. M. Chiarandini, T. Stützle, and K. S. Larsen. Colour reassignment in tabu search for the graph set t-colouring problem. In F. Almeida, M. Blesa, C. Blum, J. M. Moreno, M. Pérez, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics*, volume 4030 of *Lecture Notes in Computer Science*, pages 162–177. Springer-Verlag, Berlin, Germany, 2006.
18. R. K. Congram. *Polynomially Searchable Exponential Neighbourhoods for Sequencing Problems in Combinatorial Optimization*. PhD thesis, Southampton University, Faculty of Mathematical Studies, Southampton, UK, 2000.
19. R. K. Congram, C. N. Potts, and S. van de Velde. An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 14(1):52–67, 2002.
20. W. J. Conover. *Practical Nonparametric Statistics*. John Wiley & Sons, New York, NY, USA, third edition, 1999.
21. D. de Werra. An introduction to timetabling. *European Journal of Operational Research*, 19(2):151–162, 1985.
22. R. Dorne and J. K. Hao. Tabu search for graph coloring, T-colorings and set T-colorings. In S. Voß, S. Martello, I. H. Osman, and C. Roucairol, editors, *Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 77–92. Kluwer Academic Publishers, Boston, MA, USA, 1999.
23. I. Dumitrescu. *Constrained path and cycle problems*. PhD thesis, The University of Melbourne, Melbourne, Australia, 2002.
24. M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98:23–47, 2003.
25. C. Fleurent and J. Ferland. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 63:437–464, 1996.
26. P. Galinier and J. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999.
27. A. Gamst. Some lower bounds for a class of frequency assignment problems. *IEEE Transactions of Vehicular Technology*, 35(1):8–14, 1986.
28. B. Gendron, A. Hertz, and P. St-Louis. On edge orienting methods for graph coloring. *Journal of Combinatorial Optimization*, 13(2):163–178, 2007.
29. C. A. Glass and Adam Prügel-Bennett. A polynomially searchable exponential neighbourhood for graph colouring. *Journal of the Operational Research Society*, 56(3):324–330, 2005.

30. F. Glover. Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics*, 65(1–3):223–253, 1996.
31. F. Glover. Tabu search and adaptive memory programming – advances, applications and challenges. In R. S. Barr, R. V. Helgason, and J. L. Kennington, editors, *Interfaces in Computer Science and Operations Research: Advances in Metaheuristics, Optimization, and Stochastic Modeling Technologies*, pages 1–75. Kluwer Academic Publishers, Boston, MA, USA, 1996.
32. J. L. González-Velarde and M. Laguna. Tabu search with simple ejection chains for coloring graphs. *Annals of Operations Research*, 117(1-4):165–174, 2002.
33. A. Grosso, F. Della Croce, and R. Tadei. An enhanced dynasearch neighborhood for the single-machine total weighted tardiness scheduling problem. *Operations Research Letters*, 32(1):68–72, 2004.
34. G. Gutin and A. Yeo. Small diameter neighbourhood graphs for the traveling salesman problem: at most four moves from tour to tour. *Computers & OR*, 26(4):321–327, 1999.
35. W. K. Hale. Frequency assignment: Theory and applications. *Proceedings of the IEEE*, 68(12):1497–1514, 1980.
36. A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4):345–351, 1987.
37. H. H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2004.
38. T. R. Jensen and B. Toft. *Graph Coloring Problems*. John Wiley & Sons, New York, NY, USA, 1994.
39. D. S. Johnson and L. A. McGeoch. The traveling salesman problem: A case study in local optimization. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley & Sons, Chichester, UK, 1997.
40. R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, USA, 1972.
41. B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technology Journal*, 49:213–219, 1970.
42. D. E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 3 – Generating All Combinations and Partitions*. Addison Wesley, third edition, 2005.
43. F. T. Leighton. A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, 84(6):489–506, 1979.
44. A. Lim, Y. Zhu, Q. Lou, and B. Rodrigues. Heuristic methods for graph coloring problems. In *SAC'05: Proceedings of the 2005 ACM Symposium on Applied Computing*, pages 933–939, New York, NY, USA, 2005. ACM Press.
45. S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21(2):498–516, 1973.
46. H. R. Lourenço, O. Martin, and T. Stützle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 321–353. Kluwer Academic Publishers, Norwell, MA, USA, 2002.
47. P. Merz and B. Freisleben. Greedy and local search heuristics for the unconstrained binary quadratic programming problem. *Journal of Heuristics*, 8(2):197–213, 2002.

48. D. Neto. *Efficient Cluster Compensation for Lin-Kernighan Heuristics*. PhD thesis, University of Toronto, Department of Computer Science, Toronto, Canada, 1999.
49. C. N. Potts and S. van de Velde. Dynasearch: Iterative local improvement by dynamic programming; part I, the traveling salesman problem. Technical Report LPOM–9511, Faculty of Mechanical Engineering, University of Twente, Enschede, The Netherlands, 1995.
50. F. S. Roberts. T-colorings of graphs: Recent results and open problems. *Discrete Mathematics*, 93(2-3):229–245, 1991.
51. P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In M. J. Maher and J.-F. Puget, editors, *Proceedings of Principles and Practice of Constraint Programming*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431. Springer-Verlag, Berlin, Germany, 1998.
52. B. A. Tesman. Set T-colorings. *Congressus Numerantium*, 77:229–242, 1990.
53. P. M. Thompson and J. B. Orlin. The theory of cycle transfers. Working Paper No. OR 200-89, 1989.
54. P. M. Thompson and H. N. Psaraftis. Cyclic transfer algorithm for multivehicle routing and scheduling problems. *Operations Research*, 41:935–946, 1993.
55. M. A. Trick and H. Yildiz. A large neighborhood search heuristic for graph coloring. In P. Van Hentenryck and L. Wolsey, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 4510 of *Lecture Notes in Computer Science*, pages 346–360. Springer-Verlag, Berlin, Germany, 2007.
56. G. J. Woeginger V. G. Deineko. A study of exponential neighborhoods for the travelling salesman problem and for the quadratic assignment problem. *Mathematical Programming*, 87(3):519–542, 2000.
57. M. Yagiura and T. Ibaraki. Efficient 2 and 3-flip neighborhood search algorithms for the MAX SAT: Experimental evaluation. *Journal of Heuristics*, 7(5):423–442, 2001.
58. M. Yagiura, T. Ibaraki, and F. Glover. An ejection chain approach for the generalized assignment problem. *INFORMS Journal on Computing*, 16(2):133–151, 2004.
59. A. Zymolka, A. M. C. A. Koster, and R. Wessäly. Transparent optical network design with sparse wavelength conversion. In *Proceedings of the 7th IFIP Working Conference on Optical Network Design & Modelling*, pages 61–80, Budapest, Hungary, 2003.