Heuristic Algorithms And Learning Techniques: Applications to the Graph Coloring Problem

PhD Thesis Draft

—Daniel Cosmin Porumbel—

Contents

General Introduction

Context

The problems studied in this thesis deal with combinatorial optimization and heuristic algorithms. One of the well known drawbacks of heuristic algorithms is related to their difficulty of getting out of local optima of low quality compared to the global optimum. More generally, we observe that the heuristic strategies often lack a global vision; it seems rather difficult to integrate a form of "self-awareness" in a search process. For example, certain local search algorithm can be likened to "find the top of Mount Everest in a thick fog while suffering from amnesia" [Russell and Norvig, 2002].

This thesis is devoted to developing strategies for dealing with such type of difficulties. For illustration, it is possible to reduce the "amnesia" effects by collecting search space information while optimizing, in order to guide more effectively the subsequent exploration. A promising research direction consists of combining heuristics with knowledge discovery and learning techniques. Knowledge discovery concerns "the non trivial extraction of implicit, previously unknown, and potentially useful information from data"[Frawley *et al.*, 1992]. Such objectives are not very different from those of the researcher in optimization that is faced with algorithms exploring enormous search spaces. Our computational framework is that of the graph coloring problem.

Objectives

An objective the thesis has been to introduce a form of learning into heuristics in order to render the search strategy more "self-aware", or "well-informed". Even if machine learning seems to be a promising direction, there are several challenges to overcome. The most important is related to the operational and computational level: it is essential that any additional learning mechanism integrated in a search process does not introduce an important computational overhead.

An elegant way to bring us closer to the above objective is to use a distance measure between candidate solutions so as to to establish a positional navigation/orientation system in the search space. In data mining, the detection of structures often require performing comparisons between points: for example, in data classification, one needs to group in the same cluster all resembling individuals and in separated clusters the individuals that are different. From an operational viewpoint, such comparisons are often based on the concept of distance. A particular challenge in the graph coloring context has been to introduce a fast distance calculation method.

A more general challenge has been to effectively put at work a positional navigation/orientation system based on distances between candidate solutions. For example, we showed it is possible to record the trajectory of a search algorithm and to keep track of all visited candidate solutions by recording a smaller set of spheres (a sphere covers all potential solutions within a certain distance from its center). We have used such distance-based orientation principles to present both local search and evolutionary algorithms. The distance measure enables us to capture the notions of position, close and distance candidate solutions, spatial distribution, search process trajectory, etc.

All introduced algorithms have been implemented and validated on the graph coloring problem, but most ideas can be translate to other optimization problems, provided that a meaningful search space distance can be defined.

Contributions

The main contributions can roughly be classified into four directions:

- distance-based search space analysis (cartography) for creating a positional orientation system for guiding local search heuristics (for diversification and intensification);
- evolutionary algorithms with distance-based control of diversity (spacing) and with well informed crossover;
- new “well-informed” evaluation functions, tailored specifically to our graph coloring problem;
- an exact algorithm for computing the distance between partitions (or colorings).

Search Space “Cartography” and Position Guided Local Searches This project starts from a study of the spatial distribution of high-quality potential solutions in the search space. Using the partition distance function and a data mining technique (Multidimensional Scaling), we introduce the following clustering hypothesis: the high quality solutions are not randomly scattered in the search space, but rather grouped in clusters within spheres of specific diameter.

Based on this acquired information, one can monitor the evolution of the search process. We focused on the fact that a typical local search is not concerned with having a global vision over its own evolution path—and on the fact that it can loop indefinitely between only several regions. By recording the visited spheres, a “position guided” local search is able to guide itself toward as-yet-unvisited spheres, i.e. it does not revisit already-recorded spheres. Besides TS-Div, we also designed an intensification oriented algorithm (TS-Int) that investigates meticulously a limited perimeter around the proximity of a given potential solution; if a solution is situated within a certain distance from this given point, TS-Int finds it with 100% success rate. The study is reported in a full paper accepted by "Computers & Operation Research" [Porumbel *et al.*, 2010]; more discussions about the diversification part and related issues were presented at the LION "Learning in OptimizatioN" conference [Porumbel *et al.*, 2009c].

Spacing Control and Well Informed Grouping Recombination in the Evolutionary Approach This project shows how to use a distance measure so as to maintain population spacing (diversity) in evolutionary algorithms. Indeed, distance measures between potential solutions provide very strong mechanisms for keeping the individuals pairwise distanced. For instance, this algorithm (call it Evo-Div) inserts a new offspring solution into the population if and only if it is not "too close" to any existing individual. The proposed recombination is well-informed in the sense that it uses a large amount of information to select the best color classes to be inherited by the offspring. A work-in-progress 12 page article was presented at the 9th European Conference on Evolutionary Computation in Combinatorial Optimization [Porumbel *et al.*, 2009a], where it was selected among the three best paper nominees. A complete 25-page paper was submitted too [Porumbel *et al.*, 2009b].

New Well-Informed Evaluation Functions This project deals with investigating problem specific properties in order to devise new evaluation (objective) functions for graph coloring. By introducing two new such functions, a classical Tabu search can differentiate configurations not distinguished by the straightforward objective function. The resulting Tabu search algorithm (call it RCTS) can reach the best-known results for many standard DIMACS instances while remaining quite simple. The results of this study were submitted in 2007 to an international journal [Porumbel *et al.*, 2007a], and two revisions were performed in 2009. The positive influences of one of the evaluation functions on a parameter-free Steepest Descent algorithm were presented at the Artificial Evolution conference [Porumbel *et al.*, 2007b].

Exact Algorithms for Computing the Partition Distance Part of the analyses and algorithms from this thesis are based on the distance measure between colorings (or partitions). This project explores exact algorithms for computing this distance; under certain conditions, one can substantially reduce the computational complexity of the standard solution method—i.e. straightforward reduction to the assignment problem and application of the Hungarian algorithm. A faster algorithm was very useful in the first project above because the guided local search algorithms (TS-Div and TS-Int) resort to numerous distance calculations. Furthermore, we consider that a fast algorithm can also be useful in many other different applications, especially those concerned with computing distances between close partitions—the complexity of this operation can be reduced from $O(|V|+k^3)$ to $O(|V|)$. This algorithm was submitted to an international journal [Porumbel *et al.*, 2008].

Thesis plan

The manuscript is organized as follows.

– In the first chapter, we briefly outline the scope and objectives of heuristic algorithms, focusing on the limitations that are addressed by this thesis. We introduce the graph coloring problem, our experimental framework, and we establish the general setting and the basic terminology;

– The second chapter introduces a first approach to graph coloring, a basic Tabu Search (TS) algorithm that will also be useful in the next chapters. This basic TS is here enhanced with well-informed evaluation functions and with reactive Tabu list strategies, resulting in a simple-but-effective algorithm RCTS (Reinforced Coloring Tabu Search);

– Chapter 3 presents a search space analysis investigating the spatial distribution of high quality potential solutions visited sequentially by a local search. We introduce the notions of sphere and we provide evidence that the high quality configurations tend to occur relatively close to each other, clustered in spheres.

– Chapter 4 presents the two "guided" local search algorithms using the search space analysis: (i) TS-Div is oriented toward diversification and ensures that the search process does not repeatedly visit the same spheres again and again, and (ii) TS-Int is intensification-oriented and assures a thorough exploration of a limited perimeter by performing a breath-first-search enumeration of all spheres within a certain distance from a given starting point.

– In Chapter 5, we describe the evolutionary algorithm Evo–Div in which we also exploit search space properties from the previous chapters so as to ensure an appropriate population spacing. The introduced crossover is very well-informed, in the sense that it uses many gene features to select the information to be passed to offspring. The number of parents is automatically determined.

– Finally, in Chapter 6, we discuss in detail the distance between colorings, focusing on an enhanced exact algorithm for its calculation.

All contributions are presented in the chronological order of their development so as to introduce all notions in a very progressive (incremental) manner. Each chapter can be read independently, with minimum effort of cross-referencing to other chapters.

Chapter 1

Introduction

This chapter reviews classical heuristic algorithms and discusses their applicability to intractable optimization problems. We explain the reasons (the "why do we care") for studying heuristics, placing a special emphasis on the limitations that could be overcome by machine learning. In the second part we discuss our experimental framework—the graph coloring problem—and we present the best coloring algorithms introduced along the past three decades.

Contents

1.1 Intractable problems and heuristic algorithms

An *instance* of a combinatorial optimization problem is defined by a set of candidate solutions (i.e. the search space Ω) and an objective function. Solving such a problem (technically speaking, an instance of the problem) requires finding a solution qui optimizes (i.e. minimizes) the given objective function. In practice, certain combinatorial problems are considered computationally *intractable*. In terms of complexity, these problems are usually NP-Hard [Garey *et al.*, 1979]: any exact algorithm would require a prohibitive calculation time, exponential in the input size (we consider $P \neq NP$).[1]

Along the past five decades, intractable computational problems have been the focus of intensive research in exact algorithms. Indeed, extensive work has been done in several partially-overlapping communities related to algorithms, including, but not limited to: algorithms and complexity theory, mathematical optimization theory, operations research and artificial intelligence. Important progress has been made and, nowadays, there are three generic approaches [Hoos and Stützle, 2004, p. 27] to such problems:

– study classes of instances that accept effective exact algorithms;

– develop approximation algorithms that run in polynomial time;

– develop heuristic, metaheuristic and stochastic algorithms.

Regarding the first approach, it is important to notice that worst-case exponential algorithms can be remarkably efficient in practice. A representative example is the *simplex* algorithm that, although exponential on the worst case, is routinely used to solve linear programs in polynomial time. Other algorithms based on implicit enumeration (e.g. branch-and-bound, or branch-and-cut) can be very effective for practical problems of moderate size, although exponential on the worst case. It might happen that exact algorithms solve in polynomial time intractable problems of large size. However, the opposite can also happen very often: the exact algorithms can require prohibitive time for small size instances [Hao *et al.*, 1999, §2.3]. This is also the case for the graph coloring problem; nowadays, there is no exact algorithm that can color optimally random graphs with more than 80 vertices [Malaguti and Toth, in press].

In certain fortunate cases, intractable problems can be approached with polynomial *approximation algorithms* that, although not exact, are *guaranteed* to produce a solution of provable quality, e.g. within a fixed factor away from the optimal solution. While there are some NP-complete problems on which approximation algorithms have been proved (e.g. the bin-packing problem), most of them do not accept neither exact, nor approximate polynomial algorithms. In fact, for most real-life intractable problems, solutions of *guaranteed* good-enough quality can not be methodically provided in polynomial time.

[1]The complexity class P contains all decision problems that can be solved exactly in polynomial time on a deterministic machine. The NP class makes reference to "Nondeterministic Polynomial" problems that can be solved exactly in polynomial time on a nondeterministic machine. It is generally assumed that $P \neq NP$. An optimization problem $\mathcal{X}$ is NP-hard if there is a NP-complete decision problem $\mathcal{Y}$ that can be reduced to $\mathcal{X}$ in polynomial time[Garey *et al.*, 1979].

A practical approach to many intractable problems of large size is *heuristic search algorithms*—or simply, heuristics. These algorithms use reasonable resources and they are able to produce acceptable solutions, but without any theoretical guarantee. Heuristics can produce competitive results not only on well-known NP-complete problems, but on any computational problem for which exact algorithms require prohibitive time. For instance, even a polynomial running time could be too long to be practical in certain fields with thousands of variables (i.e. electronic circuit design). In practical terms, it is not always necessary to find the optimum solution, but only "good enough" solutions. Furthermore, optimal solutions provided by refined exact algorithms can often be reached *more rapidly* with basic meta-heuristics. As [Battiti *et al.*, 2008] put it, "we are condemned to live with heuristics for very long times, maybe forever".[2]

Besides computer science, heuristic algorithms are also subject of intensive research in other fields of computational domains (e.g. operations research, bio-informatics, engineering) in which one routinely needs to solve *computational optimization problems.* In this context, a problem can be seen as "finding the minimum value of an objective function f over some set of items". Most computational problems are routinely expressed in the optimization form. For example, a constraint satisfaction problem asks to decide whether or not one can assign certain values to a set of variables such that a set of constraints is satisfied. By defining $f(x)$ as the number of constraints violated by potential solution x, the problem is equivalent to determining if $\min f = 0$.

Given a computational problem, a heuristic is essentially a "search" algorithm that seeks the best solution in the space of all potential problem solutions—i.e. in the search space, also referred to as the problem space (called Ω). Assuming it is not possible to enumerate all potential solutions of a large intractable problem[3], the question is *how* to decide what part of Ω to explore/test in a limited time. For this purpose, one can use either problem-specific or general (meta) guidelines and search strategies. In this view, the goal of any heuristic is to employ the most intelligent search strategies so as to rapidly find the most promising search space areas, and to, hopefully, locate the solution.

The term meta-heuristic, coined in the paper that introduced the term Tabu search [Glover, 1986], represents a class of general-purpose heuristic algorithms that can be applied to any optimization problem; the only necessary condition is to define a potential solution encoding and an objective function. Certain meta-heuristics require a neighborhood relation (e.g. local searches) or a mutation operator (evolutionary algorithms), but, however, such components are regarded as "black boxes". A meta-heuristic is not concerned with defining them, but it only deals with operations at a higher (meta) level. The rest of this section assumes a certain familiarity with meta-heuristics; however, the algorithms in Chapters 2–6 are described in a self-contained manner.

[2]It seem difficult to assume that one day exact algorithms will be able to solve everything and heuristics will have no purpose.

[3]Notice that another approach in combinatorial optimization is represented by implicit enumeration algorithms like branch-and-bound or branch-and-cut. One should use this type of methods if the search space can be enumerated (explicitly or implicitly).

1.1.1 Machine learning for intelligent heuristics

While heuristic algorithms reached impressive success for some otherwise impossible problems,[4] they still have major drawbacks, most of them due to the lack of theoretical provable behavior. Indeed, it seems difficult to make (local or population-based) heuristics integrate a global vision over the whole search space and over their own trajectory through this space. For instance, at a given moment of the search, even a (more "global") evolutionary algorithm typically takes all decisions only by considering the *current* individuals and their neighborhoods, i.e. it seems difficult to take into account objects lying outside this "small word'. Using complex global analyses in optimization is complicated by the fact that search algorithms are usually very fast (e.g. a coloring algorithm visits millions of configurations per minute), producing too much information to process in a complex calculation method. However, data analysis algorithms could be used as they also have the objective of "revealing, in a large amount of data pieces, of organisation structures behind them"[Lerman *et al.*, 1981]. Such learning techniques can help make classical heuristics "more intelligent", "well informed".

While the approach of using learning in optimization seems to be quite young, there has been an important growing of interest over the last decade—numerous references are available in [Battiti *et al.*, 2008]. Although the goals of the learning techniques can vary widely across optimization communities, they have been already classified into four major approaches, see [Boyan *et al.*, 2000], a rich a collection of 14 studies grouped in four classes. We do not claim that this classification is absolute and without overlapping, but we consider it is still topical and inclusive of the most popular research threads:

Search spaces analysis Important insight into the problem structure can be achieved by a statistical analysis of the search space. In fact, all effective heuristic algorithms take into consideration, either explicitly or implicitly, the structure of the search space. Search space analyses may focus on, but are not limited to, the following: the form of the search space surface (e.g. convex or non-convex— in numerical optimization), indicators of search space asperity (e.g. ruggedness, smoothness, fitness distance correlation), structural similarities between local optima (e.g. identifying common "backbone" structures), or the number and spatial distribution of local optima (see also Section 3.1.1);

Learning evaluation functions This is an active research thread devoted to modeling new evaluation functions so as to better guide the heuristic through the search space. A more informative evaluation function should provide an exact assessment of the long-term effect of searching from a certain point. This is particularly useful if there are numerous potential solutions having the same value of the objective function. The Guided Local Search (meta)-heuristic [Voudouris and Tsang, 2003] constitutes a representative example of this research field; it advocates modifying the objective function when the search process settles on a local optimum;

[4]The objective of this chapter is not to make a review on the heuristic algorithms or on their results; the interested reader can check one of the books from the literature—a very good on is [Hoos and Stützle, 2004].

Learning models to generate solutions This direction focuses on exploiting information related to the visited local optima so as to provide indications about how to find better solutions. For this, one can create a model of generating new solutions with specialized operators, e.g. by fixing some variables to certain values (shared by all visited local optima), by estimating the distribution of global optima, or by determining a "likelihood" of finding the solution in a certain area. It has been argued that "genetic algorithms do precisely this sort of modeling" [Boyan *et al.*, 2000]: the population keeps a pool of best potential solutions and genetic operators try to combine them to construct better solutions. In some sense, at least for certain memetic algorithms in discrete optimization, this view is not less accurate than referring to Darwin's theory of evolution. A pedagogical overview is available in Chapter 6 of [Battiti *et al.*, 2008], where it is argued that Ant Colony Optimization and Estimation of Distribution Algorithms are also based on similar principles;

Algorithm selection and parameter tuning This research thread seeks to automate the process of choosing the appropriate heuristic for a problem, or the best parameters of a heuristic. It can be split in two sub-directions:

- **Heuristic Selection**—It is well known that certain algorithms are very effective for some types of problems, but less appropriate for others. While the selection of the best algorithm for a problem is usually based on human expertise (e.g. certain researcher consider Simulated Annealing is probably better than Tabu Search on problems with a very large neighborhood), this decision can be automatized. Furthermore, one can alternate between several heuristics, neighborhoods or search spaces during the optimization process. A representative example of this sub-direction is the "hyper heuristics" class of algorithms[5] that can be described as "heuristics to choose heuristics".
- **Reactive Parameter Tuning** Concerning this sub-direction, it is nowadays a common technique to use "reactive mechanisms" to automatically "adapt" the parameter values "on the fly", during the optimization process. The term "reactive mechanisms" refers to using an on-line "feedback" loop for tuning parameters according to the *current* state of the search. Reactive Tabu Search [Battiti and Tecchiolli, 1994] is a representative example of such concepts; however, algorithms can also react on the neighborhood, on the annealing schedule, or on the objective function—see Chapters 2–5 in [Battiti *et al.*, 2008]. This sub-direction is closely intertwined with the others because a parameter change can trigger a complete transformation of the algorithm. With such a change, one can transform Simulated Annealing into Steepest Descent or Metropolis Algorithm; the operators of genetic algorithms can also be chosen via parameter tuning [Maturana, 2009]

While the thesis touches on all the above aspects, the most important contributions are related to the first three points. Indeed, the first point is fully addressed in Chapter

[5]As of August 2009, more than 100 references are available in G. Ochoa's " Bibliography of Hyper-heuristics and Related Approaches", see `http://www.cs.nott.ac.uk/~gxo/hhbibliography.html`

3 in which we introduce a new clustering hypothesis: the high quality potential solutions tend to occur relatively close to each other, clustered in spheres of fixed diameter. While it is difficult to theoretically prove such an hypothesis, we support it with important empirical evidence. One should also be aware that other hypothesis from the literature can be easily rejected for graph coloring. For example, the "big valley" hypothesis [Boese *et al.*, 1994] assumes that the locally optimal potential solutions should be quite close to the global optimum solution. While this seems valid for the travelling salesman problem, for graph coloring, we observed that the quality is not correlated with the distance from an optimum solution; in fact, most near-optimal solutions have no global optimum in their proximity. We routinely encountered hard instances that admit several global solutions, all very different from one another.

The second research thread is touched on in Chapter 2 where we introduce two new evaluation functions for graph coloring. While the first one is specified before initializing the exploration, the second one is constructed during the search process. The idea is to give more importance to variables that often change their state so as to help the search process fix their values, and so, to induce diversification by encouraging other variables to change more often.

The third direction is addressed by the TS-Int algorithm from Section 4.3. TS-Int can be regarded as a post-optimization tool that (tries to) locate the solution in the proximity of an initial high-quality potential solution provided by a different algorithm. We empirically observed that TS-Int can reach a global optimum systematically, if the global optimum is situated within a certain distance from the initial provided high-quality coloring. In fact, TS-Int can be used to automatically investigate the proximities of several very distant local minima provided by a diversification-oriented algorithm TS-Div—it is very likely that the solution is situated in the proximity of one of the best local minima ever reached. We can say that TS-Int finds a solution based on an approximate location.

The last research thread is also touched on throughout the thesis, in different contexts. Two examples are: (i) the reactive Tabu tuning techniques in Section 2.4 and (ii) the reactive dispersion of the evolutionary algorithm (Section 5.4.2.2).

1.1.2 Collecting information (on-line) while searching

Learning techniques for optimization can also be classified into "off-line" techniques and "on-line" techniques. For example, most search space analyses are instances of "off-line" learning, in the sense that they are performed in a pre-optimization stage, before starting the main search space exploration phase. While such information can give essential guidelines for designing an effective heuristic algorithm, it is not used to take decisions *during* the exploration, e.g. to react to events during the search.

The automatic tuning of parameters is a natural example of "on-line" learning techniques, used "on the fly" during the optimization process. While an "off-line" search space investigation can use complex statistical clustering methods, an "on-line" search space analysis would be limited by strong constraints of computational complexity, i.e. it should not introduce an important slowdown in the search process. As such, the most common on-line learning techniques are usually very lightweight, often focusing only on reactions to local information or events—e.g. reactive Tabu tuning techniques (like the one in Section 2.4) are based on short history information. It seems that it is rather problematic to collect and integrate "on the fly" a large amount of global information, e.g. patterns of visited local optima, correlations between potential solution location and quality, convexity properties of the search space surface, etc.

However, we show in this thesis that a very lightweight learning procedure is enough to record the complete exploration path of a search process. The main idea is that, although the total number of visited potential solutions is too large to be recorded, a coarse grained recording can be performed by recording a smaller number of visited spheres. The notion of sphere is straightforwardly defined if one has a search space distance measure, i.e. it is the set of all potential solutions within a certain distance from a center potential solution.

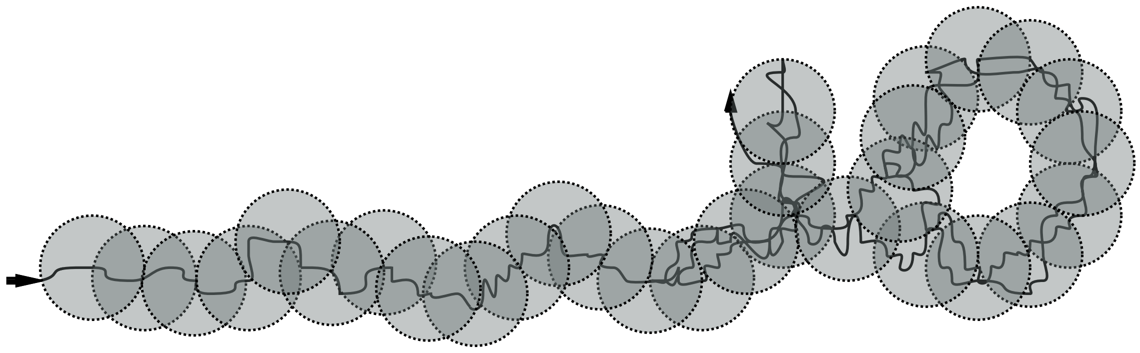

Figure 1.1: An example of a coarse-grained recording of a local search exploration path. The number of spheres is much lower than the number of visited search space points (potential solutions). The recording can be performed "on the fly" without important slowdown for the search process.

Figure 1.1 shows an example of a coarse-grained recording of an exploration path. Once equipped with a recording component, a local search algorithm can "realize" when it comes upon already-visited regions (spheres), if it is always attracted to certain regions that are repeatedly visited, whether it assures global diversification or not. The proposed recording component can be attached to any local search exploring a space that admits a distance measure, and, furthermore, it is very lightweight: it only needs to compute *a distance per iteration.* More exactly, at each iteration, it computes the distance to the center of the current sphere: if the search process is still inside the current sphere, no action is taken; otherwise, the current potential solution becomes the center of the next current sphere. One might need more computations *only if* the algorithm needs to translate this global recording into calculated decisions or actions. However, certain major actions (e.g. trigger diversification) can be decided with a reasonable amount of calculation overhead, by keeping the number of distance computations in acceptable limits (as, for example, in Section 4.2.2).

1.2 Graph coloring: framework for experimental evaluation

This section formally describes our optimization problem, the test instances and the previous methods against which we test all our new algorithms. In combinatorial optimization, graph k-coloring is no-more-no-less than a particular case of a constraint satisfaction problem. We recall that a constraint satisfaction problem is formally defined by a triple:

1. a set of variables—in our case, the set of vertices;
2. the domain of all possible values for each variable—for graph k-coloring, it is $\{1, 2, \ldots, k\}$ where k is the number of colors;
3. a set of constraints between variables—in our case, the value (or color) assigned to a variable needs to be different from the value of any "adjacent" variable (where "adjacent" refers to the edge structure). One can say that an *allDifferent* constraint is applied on certain pairs of adjacent variables.

By slightly changing this specification, numerous well-known problems can be defined. For instance, the 8 (or n) queens puzzle only requires adding some more (diagonal) constraints between the "colors"; for illustration, by replacing "queen" with "rock", one can say the puzzle becomes a coloring problem on a complete graph. Additionally, T-coloring or set coloring are also defined by generalizing the above constraints. However, in this thesis we do not employ too many techniques related to graph theory—e.g. like separating large independent sets, identifying cliques or decomposing the graph—but we actually focus on optimization techniques to minimize the number of violated constraints.

One should be aware that there is a vast body of work on graph coloring, so we only touch on the most related publications in this thesis. Indeed, in the last five years, there have been published a number of monographs, surveys and journal special issues [Galinier and Hertz, 2006; Chiarandini *et al.*, 2007; Malaguti and Toth, in press; Johnson *et al.*, 2008] with hundreds or even thousands of references to other publications.[6] Furthermore, during the last decade, there have been defended a considerable number of other PhD theses with a major part dedicated to heuristic graph coloring [Galinier, 1999; Dorne, 1998; Blöchliger, 2005; Chiarandini, 2005; Devarenne, 2007; Weinberg, 2004; Zufferey, 2002].

1.2.1 Practical applications of graph coloring

The graph (vertex) coloring problem was one of the first problems proved to be NP-complete at the beginning of the computational complexity studies [Karp, 1972]. Indeed, its intractability is infamous: except for a very limited number of special cases (trees, bipartite graphs, etc.), it can not even be approximated by a polynomial algorithm within a constant factor, unless $P = NP$—a proof for a factor of 2 is given since the 1970s [Garey *et al.*, 1979, Theorem 6.11], but newer results are also available [Lund and Yannakakis,

[6]M. Chiarandini's on-line "Bibliography on Graph-Vertex Coloring" (`www.imada.sdu.dk/~marco/gcp/`) also contains a very rich collection—about 200 references (as of 2009). One can also check J. Culberson's "Graph Coloring Bibliography" (`http://web.cs.ualberta.ca/~joe/Coloring/index.html#Color.bibliography`) that contains almost 500 references.

1994; Bellare *et al.*, 1998]. In practical terms, graph coloring has widespread applications numerous areas, as for example:

- frequency assignment in cellular networks [Hale, 1980; Gamst and Rave, 1982; De Werra and Gay, 1994; Dorne and Hao, 1995; Park and Lee, 1996];
- scheduling [Leighton, 1979; Gamache *et al.*, 2007; Lewandowski and Condon, 1996; Zufferey *et al.*, 2008; Laurent and Hao, 2009];
- register allocation in compilers [Chaitin, 1982; Lewandowski and Condon, 1996; Chow and Hennessy, 1990];
- automated timetabling [de Werra, 1985; Burke *et al.*, 1994];
- supply chain management [Lim and Wang, 2005; Glass, 2002];
- computing derivatives, Jacobian and Hessian matrices [Gebremedhin *et al.*, 2005];
- air traffic flow management [Barnier and Brisset, 2004];
- network ressource allocation [Woo *et al.*, 1991];

In fact, an exhaustive list of (direct or implicit) applications would perhaps contain dozens or hundreds of references, simply because the coloring model is very general. The above list comprises *only the most representative* applications that I am aware of. However, even more specific examples are available in the literature (e.g. train platforming, printed circuit testing, manufacturing) and the reader whose curiosity is piqued can further refer to the introduction of [Malaguti *et al.*, 2008; Lü and Hao, 2010; Blöchliger and Zufferey, 2008; Malaguti and Toth, in press]. In some sense, graph coloring is even well-known with respect to the general public via Sudoku—this game is no-more-no-less than a coloring problem with $k = 9$ colors.

1.2.1.1 Existing Heuristic Approaches

The first algorithms for graph coloring were developed in the 1960s [Welsh and Powell, 1967; Christofides, 1971; Brown, 1972]. Since then, a considerable number of new techniques have been developed and important progress has been made. While exact approaches have been tried on graph coloring [Brelaz, 1979; Mehrotra and Trick, 1996; Sewell, 1996; Ramani *et al.*, 2006; Lucet *et al.*, 2006; Méndez-Díaz and Zabala, 2006], "very few exact solution algorithms exist for the problem" [Méndez-Díaz and Zabala, 2006]. As such, heuristic methods dominate the literature of practical algorithms of *general* graph coloring.[7] Essentially, heuristic algorithms belong to three main solution approaches:

1. Sequential construction—e.g. Dsatur [Brelaz, 1979], Iterated Greedy [Culberson and Luo, 1996], RLX and XRLF [Johnson *et al.*, 1991], very fast methods but not particularly efficient;

[7]Notice that a different research thread in exact algorithms consists of investigating *particular* graph classes, *particular* types of colorings, or other related problems that are not $NP-$complete.

2. Local search
    - Tabu Search [Hertz and Werra, 1987; Fleurent and Ferland, 1996a; Dorne and Hao, 1998b; Galinier and Hao, 1999; Blöchliger and Zufferey, 2008]
    - Simulated Annealing [Chams *et al.*, 1987; Johnson *et al.*, 1991]
    - Variable Search Space or Variable Neighborhood Search [Trick and Yildiz, 2007; Avanthay *et al.*, 2003; Hertz *et al.*, 2008])
    - Iterated Local Search [Paquete and Stützle, 2002; Chiarandini and Stützle, 2002; Lü and Hao, 2009]
3. Evolutionary population-based hybrid or distributed methods [Fleurent and Ferland, 1996a; Morgenstern, 1996; Dorne and Hao, 1998a; Galinier and Hao, 1999; Galinier *et al.*, 2008; Malaguti *et al.*, 2008; Malaguti and Toth, 2008].

In this vast literature, several pieces of research are particularly important and very related to our study, requiring more attention. A first important breakthrough in heuristic graph coloring is the Tabucol (Tabu search Coloring) algorithm introduced by Hertz and Werra [Hertz and Werra, 1987] in 1987. While Tabu Search [Glover, 1986; Glover and Laguna, 1997] is commonly applied to numerous problems, a particularity of graph coloring is that the *Tabu status is not applied to potential solutions*, but to *moves* (a move is an assignment of a certain color to a certain vertex). An essential later improvement of Tabucol consists of a faster (incremental) neighborhood examination technique [Fleurent and Ferland, 1996a] that enabled calculating the objective function value of all neighbors—the original Tabucol works only with a sample of the neighborhood. Further important developments have been focused on determining an optimal length of the Tabu list, so as to decide how long to forbid a move after performing it [Dorne and Hao, 1998b; Dorne and Hao, 1998a; Galinier and Hao, 1999; Blöchliger and Zufferey, 2008; Devarenne *et al.*, 2006], see also Section 2.2.

Another important breakthrough in practical graph coloring is represented by the first crossover models oriented to groups (classes) instead of items (colors) [Dorne and Hao, 1998a; Galinier and Hao, 1999]. The Greedy Partition Crossover (GPX) enabled the corresponding paper [Galinier and Hao, 1999] to clearly dominate the tables with the best results for many years. With the introduction of this crossover in memetic algorithms (evolutionary algorithms incorporating local search), the population-based algorithms became one of the best approach to graph coloring. This is in sharp contrast with the boolean satisfiability problem (SAT) that, although generalizing k-coloring, it seems more difficult for the evolutionary approach.

Notice also that, very recently, two less-traditional local search algorithms proved surprisingly competitive [Hertz *et al.*, 2008; Blöchliger and Zufferey, 2008]. They are based on evolved neighborhood and search space structures and on partial solution representations, along with advanced Tabu tuning techniques. It is worth mentioning a “very old” paper [Morgenstern, 1996] that introduced numerous graph coloring ideas, some of them still used nowadays. This study presented several algorithms and their global results—most of them summarized in the table at page 357—still constitute a reference point for other algorithms.

In terms experimental evaluation, the second DIMACS Implementation Challenge [Johnson and Trick, 1996] collected a large set of graphs that has become a standard benchmark for testing coloring algorithms since 1996. This challenge, along with other recent international events [Johnson *et al.*, 2002; Johnson *et al.*, 2008] have boosted the worldwide research in this competitive and challenging area. Indeed, four of the best ten algorithms from the literature were published in the last year [Malaguti *et al.*, 2008; Hertz *et al.*, 2008; Blöchliger and Zufferey, 2008; Galinier *et al.*, 2008].

1.2.2 Formal definition and coloring representations

The graph coloring problem has a very simple formulation: label the vertices of a graph with the minimum number of colors (the chromatic number) such that there are no two adjacent vertices sharing the same color. Graph k-coloring is a related problem: given a connected graph $G(V, E)$ and k different colors represented by numbers $\{1, 2, \ldots, k\}$, determine whether or not there is a k-coloring (a vertex labeling/coloring using k colors) without conflicts, i.e. without edges with both ends of the same color.

Most heuristic algorithms deal with the general graph coloring problem by solving a series of increasingly-difficult k-coloring problems. This method starts with a very large initial k (e.g. $k = |V|$) and iteratively decrements k after finding a k-coloring with no conflicts. The k-coloring problem becomes more and more difficult until the algorithm fails in finding k-colorings with no conflicts. The lowest k for which the k-coloring problem was solved constitutes the best reached solution for the general coloring problem, it is an upper bound to the chromatic number of G.

Definition 1.1. *(Array representation) Given a graph G and $k \in I\!N^*$, a coloring is a function $C : V \to \{1, 2, \cdots, k\}$, encoded as an array $C = [C(1), C(2), \ldots, C(|V|)]$.*

The set of all colorings represent the potential solutions (configurations) of the search space Ω of the problem (G, k). A coloring C is said to be a legal coloring if and only if $C(i) \neq C(j)$, $\forall \{i, j\} \in E$. A legal coloring represents a *solution* to the k-coloring problem (G, k). While we also encoded this color-based representation in our programs, it is very useful to interpret a coloring as a vertex set partition.

Definition 1.2. *(Partition representation) A k-coloring $C = [C(1), C(2), \ldots, C(|V|)]$, of G is denoted as a partition $\{C^1, C^2, \ldots, C^k\}$ of V (i.e. a set of k disjoint subsets of V covering V) such that $\forall x \in V$, $x \in C^i \Leftrightarrow C(x) = i$.*

We say that C^i is the class color i induced by the coloring C, i.e. the set of vertices having color i in C. This partition definition is particularly useful to avoid symmetry issues arising from the classical color based encoding. As such, it is used in many parts of the thesis,e.g. to define a meaningful distance between colorings (see Chapter 6) or to define the crossover operator (see Section 5.3). In this context, C is a legal or conflict-free coloring (a solution) if and only no color class of C contains adjacent vertices.

Definition 1.3. *(Conflicts) Given a k-coloring C, the set of* conflicts *$CE(C)$ refers to the edges in conflict generated by C, i.e. $CE(C) = \{\{i, j\} \in E \ : \ C(i) = C(j)\}$. The set of* *vertices in conflict induced by C is given by: $CV(C) = \{i \in V | \exists j \in V$ such that $\{i, j\} \in CE\}$.*

We can call conflicting edges and conflicting vertices the elements of CE and, respectively, CV—when no confusion is possible, one can omit the argument (C).

Definition 1.4. *(Objective function for k-coloring) Given a k-coloring problem $(G_{V,E}, k)$, the objective function f assigns to each coloring the number of conflicts $|CE|$, or the conflict number. More exactly, f is denoted by:*

$$f(C) = |\{\{i, j\} \in E : C(i) = C(j)\}|, \forall C \in S \tag{1.1}$$

The k-coloring problem is a decision problem (to determine whether or not there exists a legal k-coloring) that is solved as an optimization problem: find the minimum value of $f(C)$ over all colorings C in the search space Ω. A given (G, k) problem is considered solved if and only if one finds a coloring such that $f(C) = 0$. We do not take into consideration exact algorithms (e.g. branch and bound) that could prove that $f(C) > 0, \forall C \in \Omega$, as they can not be fast enough for large instances. Indeed, the chromatic number is still unknown for many random graphs introduced almost 20 years ago—e.g. the *dsjc* graphs first presented in[Johnson *et al.*, 1991], see Section 1.2.3 below.

Note that, especially in the evolutionary part, some concepts *can* also be referred using a different notation, traditional in the corresponding community—e.g. in evolutionary computing, everybody uses “fitness function” instead of “objective function”, and so we do in Chapter 5.

1.2.3 Test data—standard DIMACS instances

The complete DIMACS competition benchmark [Johnson and Trick, 1996] is composed of 47 graphs from the following families:

1. random graphs *dsjc*X.Y with X vertices and density Y [Johnson *et al.*, 1991];
2. flat graphs *flat*X.Y generated by establishing edges only between k_p predefined classes of vertices (X= $|V|$ and Y is the chromatic number k_p) [Culberson and Luo, 1996];
3. Leighton graphs *le*X.Y with X=450 vertices and with known chromatic number Y (they have a clique of size Y) [Leighton, 1979];
4. two families of random geometrical graphs generated by picking points uniformly at random in the unit square and by joining any 2 points distances by less than a length threshold—*r*X.Y and *dsjr*X.Y [8], where $X = |V|$ and Y is the length threshold (an additional suffix ”c” in this notation signifies the complementary graph) [Johnson *et al.*, 1991];

[8]The graphs labeled *dsjr* come from the same source as the random graphs labeled *dsjc*, i.e. they are due to David S. Johnson et. al. [Johnson *et al.*, 1991] and hence the prefix *dsj*. The *r* family was generated by M. Trick using a program of C. Morgenstern, see also `mat.gsia.cmu.edu/COLOR/instances.html`

G	k^*	G	k^*	G	k^*	G	k^*
$dsjc125.1$	5	$r125.1$	5	$le450.5a$	5	$flat300.20$	20
$dsjc125.5$	17	$r125.5$	36	$le450.5b$	5	$flat300.26$	26
$dsjc125.9$	44	$r125.1c$	46	$le450.5c$	5	$flat1000.50$	50
$dsjc250.1$	8	$r250.1$	8	$le450.5d$	5	$flat1000.60$	60
$dsjc250.5$	28	$r250.1c$	64	$le450.15a$	15	$school1$	14
$dsjc250.9$	72	$r1000.1$	20	$le450.15b$	15	$school1.nsh$	14
		$dsjr500.1$	12	$le450.25a$	25		
				$le450.25b$	25		

Table 1.1: Easy DIMACS k-coloring instances. Numerous other papers report exactly the same values of the best upper bound k^* of these graphs.

5. class scheduling graphs ($school1$, $school1.nsh$) and a latin square graph ($latin\_square\_10$) [Lewandowski and Condon, 1996];

6. huge random graphs ($C2000.5$ and $C4000.5$) with up to 4 million edges [Lewandowski and Condon, 1996].

1.2.3.1 Trivial upper bounds and easy instances

Table 1.1 shows DIMACS k-coloring instances that were solved very easily with all the developed heuristics, that is, a solution was always reached in a time of seconds or minutes. Numerous other algorithms can find legal colorings with k^* colors for all these graphs, but there is no mention of a legal coloring with less colors—i.e. by using $k^* - 1$ colors, the instance probably moves into an "unSAT" side. In what follows, we concentrate only on the rest of the instances (*hard* instances), as most coloring research papers do.

1.2.3.2 Difficulty variation and results interpretation

Recall that the general coloring problem is solved through a series of k-coloring problems. All coloring algorithms—that we are aware of—reported the results in terms of the lowest k for which they solved k-coloring. Passing from a result of k to a result of $k - 1$ is not similar to a decrement in similar min-conflicts constraint satisfaction problems, i.e. *one color less* is not equivalent to *one violated constraint less*. In our case, "one violated constraint less" refers to reducing a conflict for a *fixed* k; and "one color less" refers to solving a *new* $(k - 1)$-problem—this commonly requires reducing *several* conflicts. For example, an algorithm that is able to solve a given (G, k) instance, might never find any $(k - 1)$-coloring with less than a dozen of conflicts. In this case, solving the instance with "one color less", would require reducing several conflicts; $(k - 1)-$coloring is much more difficult than k-coloring.

For this reason, in the above section, we talk about "easy instances" and not "easy graphs"; one speaks about "easy graphs" or "hard graph" only when an associated k is implicitly referred. All coloring papers, that we are aware of, always reported results in the form (G, k), i.e. they show a list of solved instances. In some sense, comparing coloring algorithms in this manner can be easily misinterpreted, e.g. for an unsolved k-coloring instance, one sees no difference between an algorithm reaching 1 conflict and an algorithm never reaching less than 5 conflicts. To our knowledge, there is no paper reporting results in terms of conflict numbers, but I personally believe that could be very useful for benchmarking purposes.

1.2.4 Reference results for performance comparison and evaluation

Throughout the thesis, we evaluate all new algorithms by comparing their results on the DIMACS benchmark with the results of the best approaches from the literature. A coloring "approach" might simply refer to a unique algorithm or to *a class of algorithms*—results are often reported by combining the performances of several algorithm versions. We briefly outline below the best approaches that we are aware of, indicating also the standard abbreviation commonly used for referencing.

Local Search Algorithms:

- MIPS—MInimal-state Processing Search, a descent algorithm with hill-climbing capabilities combined with maximum clique heuristics and greedy construction stage [Funabiki and Higashino, 2000];
- ILS—Iterated Local Search [Chiarandini and Stützle, 2002; Paquete and Stützle, 2002]), research work investigating two local search architectures with different perturbation operators;
- VNS—Variable Neighborhood Search [Avanthay *et al.*, 2003], an algorithm employing a local search and several different neighborhoods operators that are alternated so as to take the search process out of local minima;
- ALS—Adaptive Local Search [Devarenne *et al.*, 2006], a local search using a large neighborhood exploration technique with loop detection mechanisms and with a new type of Tabu list;
- PCOL—Partial Coloring Tabu search (also called PartialCol) [Blöchliger and Zufferey, 2008], a reactive Tabu search that encodes potential solutions as partial (incomplete) legal colorings;
- VSS—Variable Search Space [Hertz *et al.*, 2008], an algorithm in which the search process switches iteratively between several search spaces, each with its own encoding, objective function and neighborhood structures;

Population-based hybrid methods:

- DCNS—Distributed Coloration Neighborhood Search [Morgenstern, 1996], a large collection of algorithms and solving techniques, including population-based methods and the partial solution encoding;

- HGA—Hybrid Genetic Algorithms [Fleurent and Ferland, 1996b], a first hybridization of a coloring evolutionary algorithm with Tabu search, but with a (item) color-oriented crossover;
- CISM—Crossover by Independent Sets and Mutation Search [Dorne and Hao, 1998a], an evolutionary algorithm using "Union of Independent Sets" crossover hybridized with a special Tabu algorithm employing certain random walk moves;
- HEA—Hybrid Evolutionary Algorithms [Galinier and Hao, 1999], an evolutionary hybrid approach incorporating Tabu Search, and introducing the greedy partition-based crossover;
- AMCOL—Adaptive Memory Coloring Algorithm (also called AmaCol) [Galinier *et al.*, 2008], a hybrid evolutionary algorithm with a central memory containing independent sets that can be assembled into colorings;
- MMT—two-phase algorithm due to Malaguti, Monaci and Toth [Malaguti *et al.*, 2008], combining a hybrid evolutionary algorithm and a post-optimization phase based on column optimization;
- MCOL—Memetic Algorithm for Graph Coloring (or MemCol) [Lü and Hao, 2010], a very recent memetic algorithm with an adaptive multi-parent crossover operator and a good balance intensification/diversification;

Comparing the performance with these papers is complicated by the fact that we are dealing with different machines, different programming languages, or different implementation/programming styles. The stopping conditions can also vary quite significantly: while certain papers use machine-independent indicators (e.g. limits on the number of iterations), most recent papers impose a maximum limit of time[9], usually between one hour and several days—see exact values in Section 2.5.4 or in Section 4.4.5. However, compared to other combinatorial optimization problems, these references constitute a rich (and very up-to-date/topical) comparison base, and so, all new algorithms will always be tested against them—see Sections 2.3 (p. 37), 4.2 (p. 62) and 5.3 (p. 88). By providing as much information as possible besides the CPU time (e.g. the number of iterations), such comparisons can offer, at least for indicative purposes, a fairly representative image of the general performances of a new algorithm.

The results of all above papers are also included in an *on-line collection of best upper bounds* for graph coloring (`http://info.univ-angers.fr/pub/porumbel/graphs/`). This library has been compiled while working for the PhD project and it contains 19 references, as of July 2009. An exhaustive list of coloring heuristic algorithms would probably contain dozens or hundreds of references, but one can not perform comparisons with all of them. Several papers introducing excellent algorithmic developments are not listed here simply because of very practical reasons: certain papers

[9]As opposed to the 1980–2000 period, the CPU speed increases over time are quite moderate nowadays, and so, the computational effort associated with a given CPU time can also be more easily interpreted.

are *not* focused on reaching the best numerical performances and others have simply been published during the final months of the PhD project, when the writing of this document was more than half completed [Prestwich, 2002; Glover *et al.*, 1996; Hamiez and Hao, 2004; Hamiez and Hao, 2001; Lü and Hao, 2009; Bouziri *et al.*, 2008; Fotakis *et al.*, 2001].

Chapter 2

RCTS: A Basic Tabu Search Algorithm with New Evaluation Functions and Reactive Tabu List

This chapter starts by recalling the principles of a basic coloring algorithm (Tabucol [Hertz and Werra, 1987]) based on Tabu Search [Glover, 1986; Glover and Laguna, 1997; Hansen, 1986]. Even today, the original Tabucol algorithm, along with several variants developed since 1987, are among the most popular coloring approaches. Then, this chapter describes two contributions to improve the best-known TS approach. We devise two new evaluation functions which employ supplementary (structural or dynamic) information in addition to the classical number of conflicts. These new evaluation functions allow the search process to differentiate configurations not distinguished by the conventional evaluation function. The second contribution of this chapter concerns a simple reactive mechanism for improving the tabu list management. The resulting Reinforced Coloring TS (RCTS) algorithm can reach the best upper bounds for most DIMACS graphs while remaining still quite simple. The chapter is based on an an article submitted in November, 2007 [Porumbel *et al.*, 2007a].

Contents

2.1 Introduction

Previous studies show that among the classical metaheuristics, Tabu Search (TS) algorithms have very good experimental behavior for coloring a variety of large and hard graphs. Moreover, several more effective algorithms [Fleurent and Ferland, 1996a; Dorne and Hao, 1998a; Galinier and Hao, 1999; Galinier *et al.*, 2008; Malaguti *et al.*, 2008; Blöchliger and Zufferey, 2008; Hertz *et al.*, 2008] also use a TS algorithm as a local improvement procedure. On the other hand, one observes that existing TS coloring algorithms are basically simple and do not use any sophisticated technique. One wonders then whether the performance of a TS coloring algorithm can be further boosted by improving some of its key ingredients while still keeping the algorithm as simple as possible.

In this chapter, we propose improvements on two key components: evaluation function and tabu list management. Indeed, the evaluation function is essential because it defines the landscape of the problem and guides the search process (together with the neighborhood). The classical evaluation function for k-coloration simply counts the number of conflicts, i.e. the number of edges with both ends of the same color. The objective of a TS algorithm is to move step by step toward the minimal value of this function, to try to find an optimal k-coloring. However, the classical evaluation function can not differentiate numerous k-colorings with the same number of conflicts, even if they have different potential of leading to a solution.

For this reason, we investigate two new evaluation functions that use supplementary information in addition to the number of conflicts. The first function aims to integrate structural information related to the degrees of the conflicting vertices of the graph under consideration. The second one uses the frequencies of color changes so as to create a function based on information collected along the search. In addition to the study of these evaluation functions, we also look into a simple-but-effective reactive mechanism for improving the tabu list management. We will show that the resulting TS algorithm (hereafter referred to as RCTS - Reinforced Coloring Tabu Search) obtains very competitive results for many DIMACS graphs, although it remains reasonably simple.

The rest of the chapter is organized as follows. In Section 2.2, we introduce preliminary definitions and we present the basic TS coloring algorithm—this section is also essential in other parts of the thesis because the basic Tabu search algorithm is used in different other sections, including in the evolutionary algorithm. Section 2.3 introduces the new evaluation functions and Section 2.4 proposes a new reactive technique for tuning the tabu tenure. In Section 2.5, we perform an extensive experimental study of the reinforced coloring TS algorithm on the complete set of DIMACS graphs, followed by conclusions in the last section of the chapter.

2.2 Basic Tabu Search for k-coloring

Tabu Search was first applied to the graph coloring problem by Hertz and de Werra in 1987, leading to the well-known Tabucol algorithm [Hertz and Werra, 1987]. Tabucol showed at that time remarkable results for coloring large and hard instances. Tabucol has also inspired several other TS coloring algorithms [Fleurent and Ferland, 1996a; Dorne and Hao, 1998b; Galinier and Hao, 1999] which have achieved still better performance. Even today, these TS coloring algorithms are among the most popular and useful coloring techniques for general graph coloring. A historical presentation of Tabucol as well as a comprehensive analysis of some of the best TS algorithms can be found in a recent survey [Galinier and Hertz, 2006]. Experimental comparisons of TS coloring algorithms with other metaheuristic coloring algorithms are presented in [Hoos and Stützle, 2004; Chiarandini *et al.*, 2007].

In brief, compared with the original Tabucol algorithm, the later TS coloring algorithms make improvements in two key elements. First, in [Fleurent and Ferland, 1996a], the authors introduced an efficient incremental evaluation technique for neighborhood examination. This technique allows the algorithm to pick the best neighbor among all the neighboring solutions (Tabucol checks only a sample of the whole neighborhood). Second, more elaborate tabu-list management techniques are developed. For instance, in [Dorne and Hao, 1998b; Dorne and Hao, 1998a; Galinier and Hao, 1999], the authors introduced a conflict-based technique for dynamically tuning the tabu tenure, while some very recent papers[Blöchliger and Zufferey, 2008; Devarenne *et al.*, 2006] also discuss reactive techniques.

In the rest of this section, we present several key components of TS algorithms for k-coloring. These components are shared by several TS coloring algorithms and constitute the starting point for the study reported in this chapter.

2.2.1 Neighborhood investigation

The search space Ω of a k-coloring problem $(G(V,E),k)$ consists of all possible colorings of G; since a coloring is encoded as a color array in our programs, we obtain thus $|\Omega| = |V|^k$. A simple neighborhood function $N : \Omega \rightarrow 2^{\Omega} - \{\emptyset\}$ can be defined as follows. Given a k-coloring C, a neighboring k-coloring C' is any coloring obtained by simply changing the color $c(i)$ of a *conflicting* vertex i into a new color $c'(i)$. The transition from C to C' is referred to as a *move* and it is denoted by a couple $< i, c'(i) >$. Notice that this neighborhood focuses on conflicting vertices in order to help the search process to concentrate on influential moves and to avoid irrelevant ones (i.e. the color of a non-conflicting vertex does not need to change). Denoting the neighborhood by $N(C)$, we obtain $|N(C)| = (k-1)|CV|$, which is considerably smaller than the size of a neighborhood in which *any* vertex could change its color (i.e. $(k-1)|V|$).

To rapidly choose the best next coloring from the neighborhood, we use a $|V| \times k$ table γ in which each element $\gamma_{i,c'(i)}$ denotes the number of conflicts that vertex i would have if assigned color $c'(i)$. The variation of the conflict number that would be induced by a move $< i, c'(i) >$ is $\gamma_{i,c'(i)}$ - $\gamma_{i,c(i)}$. Since our neighborhood only deals with conflicting vertices, the best move is searched by going through all elements $\gamma_{i,c'(i)}$ with $i \in CV$ ($(k-1)|CV|$ elements). After performing move $< i, c'(i) >$, γ can be updated in $O(|V|)$ time—i.e. one needs to change touch only on columns $c(i)$ and $c'(i)$. This dynamic and incremental move evaluation technique is first employed in [Fleurent and Ferland, 1996a] and proves to be indispensable for a fast examination of the whole neighborhood.

Plateau and basin of attraction Finally, this neighborhood relation enables us to formally define another two useful notions: the *plateau* and the *basin of attraction*. A plateau is a set of configurations situated below a certain altitude level (with respect to the objective function) such that any two configurations can be connected via neighborhood moves inside the plateau. A configuration is part of basin of attraction of a local optimum if one can reach the local optimum by performing only descent (down) moves. See also [Hoos and Stützle, 2004, Ch. 5] for more formal definitions.

Definition 2.1. *(Plateau) We say subset $P \subset \Omega$ is a plateau of Ω if and only if: (i) all configurations in P have the conflict number limited by a certain quality level and (ii) for any $C_a, C_b \in P$, there exist configurations $C_1, C_2, \dots, C_n \in P$ such that: $C_1 \in N(C_a)$, $C_2 \in N(C_1)$, …, $C_n \in N(C_{n-1})$ and $C_b \in N(C_n)$.*

Definition 2.2. *(Basin of attraction) We say configuration C' belongs to the basin of attraction of local optimum C if there exist configurations $C_1, C_2, \dots, C_n \in \Omega$ such that: $C_1 \in N(C')$, $C_2 \in N(C_1)$, …, $C_n \in N(C_{n-1})$, $C \in N(C_n)$ and $f(C') \geq f(C_1) \geq f(C_2) \dots f(C_n) \geq f(C)$.*

2.2.2 Tabu list management

A tabu list is commonly regarded as a first-in-first-out structure recording recent configurations or recent moves. In our case, it is more convenient to implement it using a $|V| \times k$ table T where each element indicates a possible move. Each time a move $< i, c'(i) >$ is performed, i receives the new color $c'(i)$ and the last color of i becomes forbidden (Tabu) for the next T_ℓ (tabu tenure) iterations. In practice, each element of T records the current iteration number plus the tabu tenure T_ℓ. Consequently, in order to check whether or not a move $< i, c'(i) >$ is Tabu, $T_{i,c'(i)}$ is compared with the current iteration counter.

In our TS version, the classical Tabu tenure is $T_\ell = \alpha \cdot |CE| + \text{random}(A)$, where the values of α and A are tuned in previous papers as discussed in the experimental part (see Section 2.5.1). In this context, random(A) defines a generally-applicable tenure (fixed in relation to the quality), and $\alpha \cdot |CE|$ is used to keep certain moves Tabu for a longer time (i.e. moves associated to lower quality configurations). We also introduce a reactive component to adjust T_ℓ, as discussed in Section 2.4.

Aspiration criterion Recording moves in the tabu list forbids the underlying configuration to be re-visited for the period defined by the tabu tenure T_ℓ. In some cases, this may also prevent relevant, non visited configurations from being considered. To overcome this problem, a simple aspiration criterion is used: the tabu status of a move is removed if the move leads to a neighbor configuration which is better than the best configuration ever found so far.

2.2.3 Basic TS Formal Specification

Algorithm 2.1: General Tabu Search for k-Coloring

```
Input: graph $G$, integer $k$ (colors);
Return: $f(C^*)$ the best conflict number ever found;
Variables:
   - $C$ and $C^*$: current coloring and best coloring found so far;
   - $I$: iteration counter;
   - $T$ and $T_\ell$: tabu table and dynamic tabu tenure;
   - $\gamma$: table $|V| \times k$ of conflict number variations induced by chaque move;
Begin
  1. $I = 0$ (iteration counter);
  2. $T = 0$ (Tabu table contains no Tabu move);
  3. $C$ = any initial $k$-coloring;
  4. $C^*$ = $C$ (save the best coloring found so far);
  5. Initialize $\gamma$ (for each possible move, calculate the induced conflict
     number variation);
  6. while ($f(C) > 0$ and time/iteration limit not reached )
       - Pick the best acceptable move $< i, c'(i) >$ from the neighborhood (if
         more moves lead to a best conflict number, a random choice is taken
         using the evaluation function);
       - $T[i, c(i)] = I + T_\ell$ (set the color of $i$ tabu);
       - $c(i) = c'(i)$ (perform the move);
       - Update $\gamma$;
       - if ($f(C) < f(C^*)$) then $C^* = C$ (better coloring found);
       - $I = I + 1$;
  7. RETURN $f(C^*)$
End
```

Algorithm 2.1 shows the general TS k-coloring procedure, including all the components presented above. For a given k-coloring instance (G, k), our TS algorithm starts with an initial (random) k-coloring C. The main steps of an iteration are: (i) pick the best acceptable move by identifying in γ the smallest value $\gamma_{i,c'(i)}$, (ii) set the current color of i as tabu, (iii) execute the move and (iv) update γ accordingly. The process stops when a legal coloring is found or when a time (or iteration) limit is reached.

2.3 New Evaluation Functions

All previously-cited papers directly use the function f—the conflict number, see (1.1), p. 16—as the evaluation function. Since f only counts the number of conflicts, it has an inherent inconvenience: it makes no distinction between all configurations with the same conflict number. Indeed these configurations are equivalent for f even if they may have different potential for further improvement. In terms of the fitness landscape, f may generate large plateaus of completely equivalent colorings [Hertz *et al.*, 1994].

To overcome this difficulty, we propose to enrich f with additional information which can distinguish configurations equivalent in terms of f. We introduce a heuristic function $h : \Omega \to [0, 1)$ and combine it with f by the following simple linear form, leading to a new evaluation function $\widetilde{f}$:

$$\widetilde{f}(C) = f(C) - h(C) \tag{2.1}$$

Let us consider two configurations C_1 and C_2 such that $f(C_1) = f(C_2) \neq 0$. Ideally, we should have $\widetilde{f}(C_1) < \widetilde{f}(C_2)$ if the probability to reach a conflict-free k-coloring is greater when the search runs from C_1 than when it runs from C_2. Unfortunately, due to the complexity of the fitness landscape, this probability is unknown, and the computation of a robust estimation is very difficult. For that reason we here restrict ourselves to local information easy to calculate. We propose two different heuristics.

The first one only depends on structural information of the graph: the distribution of the *conflict degrees of the vertices* (see below). The second one is based on a learning mechanism using a long-term memory: it depends on the distribution of the *frequencies of color changes* on the vertex set during a first stage of the search process.

2.3.1 A degree-based evaluation Function

Specific Definitions Besides the definitions from Section 1.2.2, let us introduce certain new definitions used only in this section.

Definition 2.3. *(Conflicting degree of a vertex) Let i be a vertex, δ_i its degree, and C a configuration. We define $CV_i = \{j \in V | \{i, j\} \in CE(C)\}$. We call $\frac{|CV_i|}{\delta_i}$ the conflicting degree of i under C.*

It is easy to see that $0 \leq |CV_i| \leq \delta_i \ \forall i \in V$; the minimal value $|CV_i| = 0$ is reached for non-conflicting vertices, while $|CV_i| = \delta_i$ indicates that vertex i is conflicting with all its neighbors. Moreover, the following relation holds for any coloring: $2|CE| = \sum_{i \in V} |CV_i|$, where CE is the set of conflicts (see also definition 1.3 in Section 1.2.2).

To motivate the new function, let us first consider the example on Figure 2.1 which depicts two configurations C_1 and C_2 for a 3-coloring on a very simple graph. The edges in conflict are respectively $\{a, b\}$ for C_1 and $\{a, c\}$ for C_2. Consequently, $|CE(C_1)| = |CE(C_2)| = 1$ and the two configurations are thus equivalent for f. However, it is easier to solve the 3-coloring problem from C_1 than from C_2.

Indeed, since the degree of b is small, one can assign to b a color not used by its neighbors (*i.e.* black or white) to solve the $\{a, b\}$ conflict on C_1. That can be done in

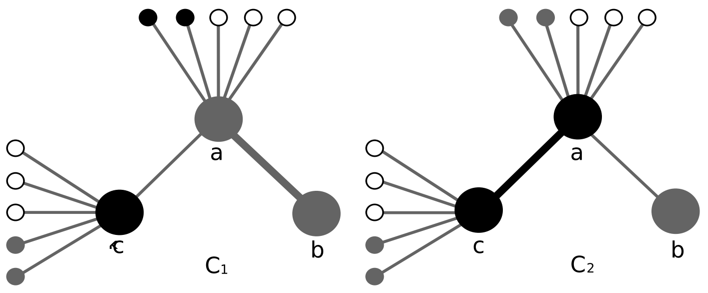

Figure 2.1: Two 3-colorings C_1 and C_2 with one conflict. The conflict is marked in with a thicker line; it is easier to solve the gray one (C_1, left) than to solve the black one (C_2, right) even if $f(C_1) = f(C_2) = 1$.

one step and it does not introduce any other conflicts. Solving the $\{a, c\}$ conflict on C_2 is more difficult because any color change on vertex a or c would perturb one of its more numerous neighbors. Intuitively, the more neighbors a vertex has, the more difficult it is to change its color without perturbing the rest of the configuration.

More generally, we can use the degree to define a penalty term: an edge with high-degree ends should count as more "heavy" in the evaluation function. In order to take all the conflicting vertices into account, we use the following heuristic h_1 to define the penalty associated to a k-coloring C:

$$h_1(C) = \frac{1}{2|E|} \sum_{i \in CV} \frac{|CV_i|}{\delta_i} \tag{2.2}$$

In this context, we see that $h_1(C)$ gives the total of the conflicting degrees of all the conflicting vertices of C. Our first degree-based evaluation function $\widetilde{f_1}$ can now be defined as follows:

$$\widetilde{f_1}\,(C) = f\,(C) - h_1\,(C) \tag{2.3}$$

The only role of the $\frac{1}{2|E|}$ coefficient in Equation 2.2 is to keep the value of h_1 in $[0, 1)$. Therefore, $\widetilde{f_1}$ preserves the f ordering: $\widetilde{f_1}\,(C) < \widetilde{f_1}\,(C')$ whenever $f\,(C) < f\,(C')$. The role of the new evaluation function is to help the search process choose between neighbors with the same conflict number, and not to introduce penalties that outweigh the number of conflicts.

Certain implementation details might be useful. At each iteration, the basic Tabu Search has to choose *at random* a neighbor that minimizes the evaluation function. Implementing such a "random choice" is not straightforward, because it is quite inefficient to simply collect and record all admissible neighbors and then choose one. We propose assigning a random value $r_{C'}$ in $[0, 1]$ for each neighbor C', at the moment when C' is discovered; in the end, the C' with the highest $r_{C'}$ value is selected. This procedure does not require recording all neighbors at each step, and it introduces no bias. Toward the end of the thesis, we realized that better experimental results can be obtained by coupling the new evaluations functions in this procedure. Let $h_{\min}$ and $h_{\max}$ be the lowest and highest value ever reached by function h. The best experimental results can be obtained by picking the $r_{C'}$ value in $\left[0, \frac{h(C')-h_{\min}}{h_{\max}-h_{\min}}\right]$.

2.3.2 Using dynamic information to construct other functions

In this section, we propose a second heuristic h_2 which takes into account information collected during the search process. The considered information is the color changes per vertex. Basically, if a vertex changes its color frequently, it should be penalized in order to discourage the vertex to still change its color.

More precisely, let us consider a first stage of the search with the basic evaluation function f. For each vertex i, we compute a frequency coefficient $freq\,(i)$ which is a scaling of the number of color changes applied on i during the first stage. The heuristic h_2 and the second new evaluation function can now be defined as follows:

$$\widetilde{f_2}\,(C) = f\,(C) - h_2\,(C) = f\,(C) - \sum_{i \in CV} |CV_i| \cdot \frac{1}{freq(i)} \tag{2.4}$$

We can say that the degree δ_i is replaced by a frequency coefficient. Given two colorings with the same conflict number, the evaluation function $\widetilde{f_2}$ prefers the one whose conflicting vertices have smaller frequencies of color change—this implies that it prefers solving the conflicting vertices of high color change frequency before the others. In practical terms, one can say that the vertices with high frequencies of color changes are considered more critical. The new function encourages the search process to solve these vertices before those with small frequencies of color change. We observed that $\widetilde{f_2}$ results in more frequent color changes on vertices that were relatively fixed during the first stage; it has a natural diversification effect.

2.3.3 Related research and complexity remarks

2.3.3.1 Related Research

One can find several pieces of research on graph coloring using ideas related to ours. A heuristic function was introduced in [Glover *et al.*, 1996], in which the authors propose to wait to color the small degree dependent vertices until all other vertices are colored. The Impasse coloring algorithm [Morgenstern, 1996] is based on a different encoding (partial colorings), but it also intends to first color high degree vertices, leaving more vertices of smaller degree in the uncolored class. This idea was also used in [Malaguti *et al.*, 2008].

Regarding the function $\widetilde{f_2}$, we note that in [Devarenne *et al.*, 2006], the vertices "frequently changed during the $n = N/2$ last iterations" are also considered for obtaining diversification. Other different evaluation functions can also be found in the literature, i.e. Johnson et. al. proposed for simulated annealing [Johnson *et al.*, 1991] the function $\hat{f}_{\text{dsjc}} = -\sum_{i=1}^{k} |C_i| + \sum_{i=1}^{k} 2|C_i||CE_i|$, where C_i is the set of vertices having color i and CE_i is the set of conflicts of color i.

2.3.3.2 Reformulation and computational complexity

An advantage of our new functions is that they introduce negligible computational overhead. Denoting any of the two heuristics by h (see Equation 2.1), one can compute h using the formula:

$$h(C) = \sum_{i \in V} |CV_i| \cdot h_i = \sum_{\{i,j\} \in CE} (h_i + h_j)$$

where h_i is the penalty associated with vertex i, i.e. $h_i = \frac{1}{2|E| \cdot \delta_i}$ for the first function, or $h_i = \frac{1}{freq(i)}$ for the second. Consequently, since $f(C) = \sum\limits_{\{i,j\} \in CE} 1$, one can calculate the value of any new function $\hat{f}(C)$ by summing up $1 - (h_i + h_j)$ for all conflicts $\{i, j\}$. Before starting the exploration, we construct a table E so that the $\hat{f} = \sum_{\{i,j\} \in CE} E_{ij}$. The only computational difference between the three evaluation functions is the initial value of E: for the classical function, E_{ij} is always 1 while, for example, for $\widetilde{f}_2$, it becomes $1 - \frac{1}{freq(i)} - \frac{1}{freq(j)}$. Regarding the first function, it is imporatnt to see that it can also be written as:

$$\widetilde{f}_1(C) = \sum_{\{i,j\} \in CE(C)} \left(1 - \frac{1}{2|E|\delta_i} - \frac{1}{2|E|\delta_j}\right) \tag{2.5}$$

2.4 A simple reactive technique for tuning the Tabu tenure

It is well known that tabu list must be managed with care. In Section 2.2.2, a dynamic mechanism is presented for tuning tabu tenure: $T_\ell = \alpha \cdot |CE| + random(A)$. In this section, we reinforce this mechanism by introducing a simple-but-effective reactive technique.

Let us consider a plateau with n configurations $C_1, C_2, \ldots, C_n$ having the same conflict number $|CE|$. A tabu list of length $n-1$ is not sufficient to break a cycle of length n: $C_1 \to C_2 \cdots \to C_n \to C_1$. Since our classical tabu tenure is bounded (i.e. T_ℓ is always less than a large n), numerical experiments show that it may not be sufficient to avoid certain loops in larger-than-average plateaux.

To overcome this difficulty, we consider a reactive tabu list: when the conflict number stays constant during a given number of iterations $M_{\max}$, we increment the length of the tabu tenure for all the subsequent iterations. In other words, when we observe $M_{\max}$ consecutive transitions $C_1 \to C_2 ... \to C_{M_{\max}}$ such that $f(C_1) = f(C_2) = \cdots = f(C_{M_{\max}})$, the tabu tenure becomes $T_\ell + 1$ for the forthcoming iterations. If the conflict number still stays constant for another $M_{\max}$ iterations, the tabu tenure becomes $T_\ell + 2$; after another $M_{\max}$ iterations it becomes $T_\ell + 3$, etc. The tabu tenure is thus continually incremented as long as the conflict number remains constant. Since we reset it to the original T_ℓ value only when the conflict number changes again, we guarantee that, sooner or later, the tabu list is increased to a value that can break a cycle of any length.

Let us remark that using a large tabu list during all the process could have a negative effect; outside large plateaux, it could encourage the algorithm to leave promising regions too early. Here, the algorithm learns from its own search evolution, and resorts to a larger tabu list only when it is necessary. This simple mechanism enabled TS to solve important looping problems on plateaus without affecting the algorithm performance outside plateaus. Finally, our Tabu list length can be express by the formula:

$$T_\ell = \alpha|CE| + random(A) + \frac{M_{\text{cst}}}{M_{\text{max}}}, \tag{2.6}$$

where M_{cst} is the number of the last TS iterations with *constant* conflict number.

To relate the present contribution to earlier research, this Tabu setting strategy belongs to the family of reactive search algorithms [Battiti *et al.*, 2008]; Reactive Tabu setting techniques based on looping detection were independently studied in other recent graph coloring papers [Blöchliger and Zufferey, 2008; Devarenne *et al.*, 2006]. The most similar approach [Blöchliger and Zufferey, 2008] considers that the search process is trapped if the objective function fluctuation stays a long period bellow a certain threshold; this threshold is set by a separate tuning phase along with two other parameters of the reactive component. This reactive scheme (called the FOO scheme) can also be regarded as a generalized, more complex version of our Tabu tenure procedure. The authors of [Devarenne *et al.*, 2006] consider some different ideas to detect looping, focusing on the identification of vertices causing loops. More details about the practical influence of our reactive tuning are given in the experimental part (Section 2.5.3).

2.5 Experiments and Discussions

In this section, we report empirical results of the reinforced coloring TS algorithm on the complete set of DIMACS coloring benchmarks. The main purpose is to assess the influence of the two new evaluation functions as well as the reactive Tabu management technique.

All DIMACS graphs were introduced in Section 1.2.3 (page 16). Recall that the coloring algorithm actually tries to solve DIMACS instances (G, k) (i.e. a k is provided for each graph) and that difficulty level of finding conflict-free k-colorings can vary from trivial to very hard. Indeed, all *easy instances* from Section 1.2.3.1 were solved by RCTS with 100% success rate in times of minutes. In what follows, we concentrate only on the rest of the instances (*hard* instances), as most coloring research papers do.

2.5.1 Experimental conditions and parameters

Recall that RCTS requires only three parameters: A, α and M_{max}. The first two are inherited from the previous TS versions, the last one is used by the new reactive tabu tenure (Section 2.4):

- A and α—they are utilized to compute the tabu tenure ($T_\ell = \alpha \cdot |CE| + random(A)$, see Section 2.2.2). We commonly set these parameters to values already reported in the literature [Galinier and Hao, 1999; Dorne and Hao, 1998b]: $A = 10$ and $\alpha = 0.6$. Our own calibration experiments confirm that this combination constitutes a good setting for our TS algorithm and consequently will be used for all our experiments.

- $M_{\max}$—after $P_{\max}$ iterations with no conflict number variation, one assumes the search process is blocked, and so, the reative component is activated (see Section 2.4). In our experiments, $M_{\max}$ is always fixed to 1000 for all instances, but there are many safe values one can assign to $M_{\max}$. We empirically observed that each time the conflict number $|CE|$ stayed constant for 1000 iterations, $|CE|$ remained so forever (with the classical tunning). If larger $M_{\max}$ values are used, the only difference is that the reactive reaction is less prompt. Smaller $M_{\max}$ values can trigger reactive reactions more often than necessary, thus influencing the search process without reason.

2.5.2 Influence of the Evaluation Function

In this section, we present experimental evidence to assess the usefulness of the new evaluation functions.

2.5.2.1 Influence on the TS algorithm

A convenient method to appreciate the influence of the evaluation function consists in inspecting the running profile of the same algorithm with different functions. We show that the new evaluation function usually allows the search process to visit more high-quality configurations in average.

Figure 2.2 depicts a typical evolution of the conflict number for RCTS with f and $\widetilde{f_1}$ on a random graph along the first 25.000 iterations. The profile of $\widetilde{f_2}$ is not shown here since it usually overlaps f. This experiment shows that $\widetilde{f_1}-$RCTS stays most of the time at lower conflict numbers than $f-$RCTS. Such a profile is observed on numerous hard graphs, providing preliminary evidence that $\widetilde{f_1}$-RCTS leads most of the time to higher quality configurations than f-RCTS does.

Table 2.1 formally reinforces this evidence by showing the average conflict numbers along 1.000.000 iterations with all three functions on some representative graphs from different families. The differences between these averages were confirmed by a statistical test. We considered the null hypothesis that the average of the conflict numbers obtained with $\widetilde{f_1}$ (or $\widetilde{f_2}$ respectively) is equal to the average obtained with f. Using a very confident level of significance of $\alpha = 0.1\%$, this hypothesis was rejected in most of the cases, confirming that most reported differences are statistically significant—see the last two columns of Table 2.1.

Table 2.1 shows that the average conflict number is always smaller for $\widetilde{f_1}$ than for $\widetilde{f_c}$ and, except for one graph, this quality difference was always confirmed by the statistical test. In the best cases, the conflict number values provided by $\widetilde{f_1}$-RCTS can be even half of those provided by f-RCTS. The second function $\widetilde{f_2}$ also shows an improvement on two thirds of instances, but with a smaller amplitude.

For the coloring problem, there is no theoretical guarantee that the probability to reach an optimal solution is strictly correlated with the average quality of the solutions visited along the search. However, especially for local searches, almost all algorithms try to guide the search process toward higher quality configurations.

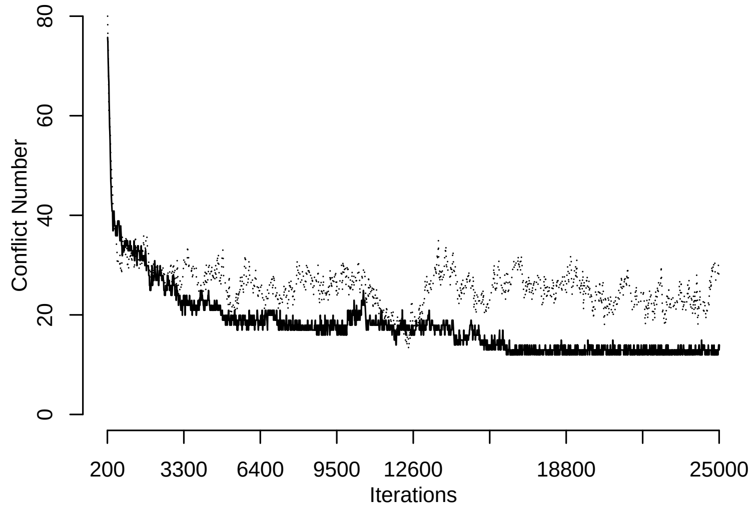

Figure 2.2: A typical evolution of the conflict number (running profile) with f (dotted line) and $\widetilde{f_1}$ (continuous line) from iteration 200 to iteration 25.000. $\widetilde{f_1}$ allows the search to visit more configurations of higher quality (fewer conflicts).

2.5.2.2 The evaluation function with empty Tabu list

In order to be sure that the general idea of a more discriminant evaluation function is not restricted to the TS algorithm or to a specific tabu tenure, we carry out a second experiment with a more neutral parameter-free steepest descent (SD) algorithm. Technically, SD is the same Tabu Search from Section 2.2, but without tabu list (i.e. $T_\ell = 0$). It starts from a random coloring and it is only guided by the evaluation function, i.e. it iteratively chooses the best neighbor according to this function. A very small number of iterations (i.e. less than 1000) is enough to descend to a first local optima where SD stops. Since a learning stage of several hundred iterations would be not sufficient for $\widetilde{f_2}$, this experiment is performed only with $\widetilde{f_1}$ and f.

We performed 1000 independent SD runs (each from a different start random configuration) with $\widetilde{f_1}$ and f and we compared the number of conflicts of the k-colorings finally obtained with each function. Figure 2.3 shows the distributions of these conflict numbers (i.e. the x-axis is the conflict number and the y-axis is the frequency): $\widetilde{f_1}$ is clearly much better because even its worst performance is often better than the best one with f. These distributions were observed for several tested graphs from all families. Note that the new function allows even this rudimentary SD algorithm to reach a solution for $\{le450\_25a, 25\}$, while the same SD with f leaves at least 10 conflicts.

Graph	k	Average Number of Conflicts			Stat. Confirmation	
		$\widetilde{f}_1$	$\widetilde{f}_2$	f	$[\widetilde{f}_1] \neq [f]$	$[\widetilde{f}_2] \neq [f]$
$dsjc250.5$	28	8.138	7.133	9.878	Yes	Yes
$dsjc500.5$	48	24.9	28.5	26.6	Yes	Yes
$dsjc1000.5$	87	35	32.2	37.7	Yes	Yes
$dsjr500.5$	122	5.106	8.875	10.46	Yes	Yes
$r1000.5$	234	16.02	25.76	29.09	Yes	Yes
$le450\_25c$	25	9.38	9.524	12.97	Yes	Yes
$le450\_25d$	25	9.183	13.94	13.74	Yes	No
$flat300\_28\_0$	30	21.63	23.87	22.13	No	No
$flat1000\_76\_0$	86	30.39	29.02	32.04	Yes	Yes

Table 2.1: Average conflict number along the first 1 million iterations with all three functions. In most cases, the modified functions, in particular $\widetilde{f}_1$, leads to colorings with (statistically) fewer number of conflicts.

2.5.2.3 Performances and instance characteristics

We also remarked that the favorable influence of the new function $\widetilde{f}_1$ is more visible on some instances than on others. Generally speaking, the best improvement is seen on the *most difficult* instances and on *some specific* classes of graph. A very good example of impressive performance is given by the random geometrical graphs ($dsjr$X.Y and rX.Y) in which $\widetilde{f}_1$ strongly dominates f with no exception—see Table 2.1 and also Table 2.2 with complete results.

This performance variation is due to the structure of the graphs, more exactly to the degree variation. For example, in the case of geometrical graphs (e.g. $dsjr500.5$), the maximum degree can be with an order of magnitude higher than the minimum degree and this makes any degree-based differentiation very effective. Indeed, the average graph class effectiveness of $\widetilde{f}_1$ can be ranked according to the degree variation, from the highest to the lowest: random geometrical graphs, Leighton graphs, random graphs, flat graphs. An extreme case is the Latin square graph which is regular (i.e. the degree is constant), and the new degree-based evaluation function brings no new distinction between vertices.

2.5.3 Influence of the Reactive Tabu List

To evaluate the influence of the new reactive tabu tenure, we analyzed and compared the classical tabu tuning (Section 2.2.2) with the reactive tabu tuning. Recall that the objective of the reactive part is to avoid looping on k-colorings with the same conflict number. Here, we consider that the search process is stuck from the moment when the conflict number does no longer variate. Using several representative graphs from each important

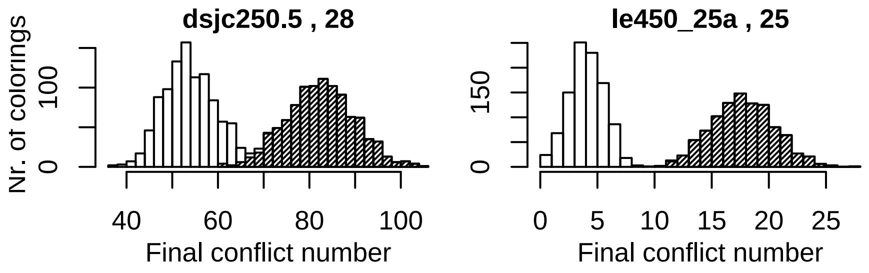

Figure 2.3: Histograms of best conflict numbers independently reached by 1000 steepest descents using $\widetilde{f_1}$ (simple bars) and f (shading lines). SD always leads to smaller number of conflicts using $\widetilde{f_1}$ than using f.

family we performed between 20 and 100 executions of the TS algorithm equipped with the classical tabu tuning and we counted how many of them got stuck before reaching 20 million iterations. The conclusion of this experiment was very clear: more than 90% of the executions got stuck looping on a plateau long before reaching 20 million iterations.

By introducing the reactive part, these looping problems are solved and the TS algorithm can successfully escape most plateaus. Thus, RCTS can effectively take profit from higher running times—without reactive tuning, there would be no use to run RCTS more than several million iterations (regardless of the evaluation function). A major positive point of our reactive scheme is the simplicity: only one parameter $M_{\max}$ is needed and it can easily be assigned a safe value—i.e. any value X satisfying the property that if the conflict number $|CE|$ stays constant for X iterations, $|CE|$ remains constant forever. We empirically observed the reactive component is triggered only a few times in millions of iterations (on plateaus that would otherwise block the process), so that the interference with other algorithm components is minimal.

2.5.4 Complete RCTS Results on All Hard Instances

In this section, we present the complete results of RCTS with all three evaluation functions for all hard instances. For each pair $\{G, k\}$, we perform 10 executions and we report the success rate, as well as the average computing effort for finding a conflict-free coloring. The stopping condition is to reach 10 hours on a 2.8 GHz Xeon processor using the C++ programming language compiled with the -O2 optimization option (gcc version 4.1.2 under Linux).

Notice that in the coloring literature, it is a common practice to run a coloring algorithm for at least several hours. For instance, a cutoff of 10 hours is also used in [Hertz *et al.*, 2008, Table 5], [Blöchliger and Zufferey, 2008, Table 6] (with a the number of iterations of up to 2000 million). Similarly, another recent paper uses a time limit of more than 10 hours (e.g. 40000 seconds in [Malaguti *et al.*, 2008, p. 310]). We find also papers using even more than several days (e.g. [Dorne and Hao, 1998a; Morgenstern, 1996]).

It is important to observe that increasing the running time will not lead to improved performance if the search algorithm gets stuck in a local optimum. On the contrary, if an algorithm is able to obtain better results when its running time is increased, this should be considered as a favorable feature of the algorithm because this indicates the search is not blocked by local optima.

Graph	k	$\widetilde{f}_1$-RCTS			f-RCTS			$\widetilde{f}_2$-RCTS		
		#Hits [/ 10]	#Iters [10^6]	Time [h]	#Hits [/ 10]	#Iters [10^6]	Time [h]	#Hits [/ 10]	#Iters [10^6]	Time [h]
$dsjc500.1$	12	10/10	96	$< \frac{1}{4}$	10/10	64	$< \frac{1}{4}$	10/10	64	$< \frac{1}{4}$
$dsjc500.5$	48	1/10	1352	6.33	0/10	—	—	0/10	—	—
$dsjc500.9$	126	9/10	360	1.67	10/10	457	2.4	10/10	466	2.5
$dsjc1000.1$	21	10/10	1	$< \frac{1}{4}$	10/10	2	$< \frac{1}{4}$	10/10	2	$< \frac{1}{4}$
$dsjc1000.5$	87	1/10	873	7	0/10	—	—	0/10	—	—
$dsjc1000.9$	224	4/10	420	7.5	1/10	321	5	3/10	492	7.33
$dsjr500.1c$	85	5/10	470	1.2	1/10	7	$< \frac{1}{4}$	1/10	7	$< \frac{1}{4}$
$dsjr500.5$	122	7/10	469	2.4	0/10	—	—	0/10	—	—
$r250.5$	65	10/10	99	$< \frac{1}{4}$	0/10	—	—	0/10	—	—
$r1000.5$	237	2/10	1059	7.5	0/10	—	—	0/10	—	—
$le450.15c$	16	8/10	11	$< \frac{1}{4}$	9/10	17	$< \frac{1}{4}$	9/10	17	$< \frac{1}{4}$
$le450.15d$	16	10/10	1	$< \frac{1}{4}$	10/10	< 1	$< \frac{1}{4}$	10/10	< 1	$< \frac{1}{4}$
$le450.25c$	25	9/10	621	1.1	6/10	572	1.2	6/10	203	1.2
$le450.25d$	25	9/10	937	2	2/10	1895	4.5	2/10	1895	4.5
$flat1000.76$	87	10/10	290	2.3	10/10	265	2	10/10	265	2
$flat300.28$	30	4/10	1183	4.5	6/10	538	2.33	8/10	737	3.1
$latin\_square$	100	6/10	641	3	4/10	1005	5	5/10	1141	5.5
$C2000.5$	162	4/10	237	3.63	1/10	601	9.5	2/10	477	8
$C4000.5$	305	4/10	88	4.75	2/10	85	4.5	2/10	98	5.5

Table 2.2: Detailed results of RCTS with a time limit of 10 hours for all three evaluation functions. $\widetilde{f}_1$-RCTS finds better solutions than f-RCTS on 25% of graphs and has an improved success rate on another 25% of graphs. The difference between $\widetilde{f}_2$-RCTS and f-RCTS is less pronounced.

In Table 2.2, we report detailed results of RCTS on all hard DIMACS graphs. The first two columns denote the k-COLOR instance, i.e. the graph and the number of colors k. For each evaluation function, we provide the success rate (Columns 3,6,9) and the average computation efforts used to solve the instance: the average number of iterations in millions (Columns 4, 7, 10) and the time in hours (Columns 5, 8, 11). While the number of iterations is a machine independent measure, the time is given for indication only. The symbol "-" means no solution is found within the given 10 hours.

First, we observe that $\widetilde{f}_1$-RCTS finds better k-colorings (i.e. with a smaller k than f-RCTS) fors more than 25% of the instances—i.e. 5 graphs our of 19. As also pointed out in Section 1.2.3.2, a difference of one color is commonly associated with *several* reduced conflicts; for certain graphs, we are scheptical that the number of colors can be reduced by any algorithm (see also the best algorithms in Table 2.3). Secondly, we observe that $\widetilde{f}_1$-RCTS obtains a success rate twice as good as f-RCTS for another 25% of graphs.

Thus, one can say that the new evaluation function $\widetilde{f_1}$ brings important improvement for more than half of the instances (i.e. 10 out of 19) with respect to the initial function f. We can also observe that the highest progress is reported especially on the most difficult instances, and on graph classes showing a large degree variation (see also Section 2.5.2.3).

Indeed, the best performance is obtained for the random geometrical graphs, for which $\widetilde{f_1}$-RCTS can quickly (less than one hour) discover solutions that f-RCTS can not find in 10 hours—e.g. see graph $r250.5$. Moreover, $\widetilde{f_1}$-RCTS also solves the difficult instance ($dsjr500.5$, 122) in less than two hours and with a stable success rate. Out of the best ten algorithms from the literature (see also Table 2.3), this instance was previously solved only by one (much more complex) algorithm [Malaguti *et al.*, 2008]. As discussed in Section 2.5.2.3, the performance improvement of $\widetilde{f_1}$-RCTS can also be lower for graphs in which the degree variation is very low (i.e. $flat$ graphs) and the discrimination power of $\widetilde{f_1}$ is less important.

The advantage of the evaluation function $\widetilde{f_2}$ over f is less visible. This can be explained by the fact that $\widetilde{f_2}$-RCTS diverges from f-RCTS only in the second half of the search, i.e. after 5 hours of computation. As a direct consequence, the values reported by f-RCTS and $\widetilde{f_2}$-RCTS are completely identical on all instances that never require more than 5 hours to be solved. By considering now the 4 instances that require more than 5 hours (i.e. $dsjc1000.9$ and the last three graphs), one sees that $\widetilde{f_2}$ has an improved success rate for 3 instances out of 4.

2.6 Chapter conclusions

We have presented the Reinforced Coloring TS (RCTS) algorithm which extends the previous TS coloring algorithms by introducing new evaluation functions and a reactive tabu tenure. By enriching the conventional evaluation function f based on the number of conflicts, the new functions take into account additional information related to the graph's structure (degrees of conflicting vertices) as well as dynamic knowledge learned along the search (frequencies of color changes). Moreover, the reactive tabu list management allows the algorithm to break loops and effectively take profit from additional computing time. Once equipped with these new components, the Tabu search algorithm, although quite simple, is able to find numerous best-known solutions of the DIMACS graphs.

Table 2.3 contrasts the results of RCTS with ten best algorithms from the literature—five of them are based on local search and the five others are hybrid algorithms. Even if a detailed comparison goes beyond the purpose of this chapter, the results give a general image about the relative performance of the proposed RCTS algorithm on the set of DIMACS graphs.

Evaluation functions in a general context Finally, let us comment that the proposed evaluation functions can be directly employed by other algorithms, e.g. in the local search component of the evolutionary algorithm from Chapter 5. More generally, it seems that the issue of evaluation function is somewhat overlooked until recently. We are convinced that this is a fundamental issue for metaheuristics, i.e. a carefully designed evaluation function,

Graph	χ/k^*		Local Search Algorithms					Hybrid Algorithms					
		RCTS	ILS	VNS	ALS	PCOL	VSS	DCNS	HGA	HEA	AMCOL	MMT	MCOL
			2002	2003	2008	2008	2008	1996	1996	1999	2008	2008	2010
*dsjc*500.1	?/12	12	12	–	13	12	12	–	–	–	12	12	12
*dsjc*500.5	?/48	48	49	49	50	48	48	49	49	48	48	48	48
*dsjc*500.9	?/126	126	126	–	128	126	126	–	–	–	126	127	126
*dsjc*1000.1	?/20	21	–	–	21	20	20	–	–	20	20	20	20
*dsjc*1000.5	?/83	87	89	90	89	89	88	89	84	83	84	83	83
*dsjc*1000.9	?/224 [223]	224	–	–	230	225	224	226	–	224	224	225	223
*dsjr*500.1*c*	84/85	85	–	–	–	85	85	85	85	–	86	85	85
*dsjr*500.5	122/122	122	124	–	–	126	125	123	130	–	125	122	122
*r*250.5	65/65	65	–	–	–	66	–	65	69	–	–	65	65
*r*1000.5	234/234	237	–	–	–	248	–	241	268	–	–	234	245
*flat*300.28	28/28	30	31	31	–	28	28	31	33	31	31	31	29
*flat*1000.76	76/82	87	–	89	–	88	86	89	84	83	84	82	82
*le*450.15*c*	15/15	16	15	15	–	15	15	15	15	15	15	15	15
*le*450.15*d*	15/15	16	15	15	–	15	15	15	15	–	15	15	15
*le*450.25*c*	25/25	25	26	–	–	25	26	25	25	26	26	25	25
*le*450.25*d*	25/25	25	26	–	–	25	26	25	25	–	26	25	25
latin_square	?/98	100	99	–	–	–	–	98	106	–	104	101	99
*C*2000.5	?/150 [148]	162	–	–	–	–	–	150	153	–	–	–	148
*C*4000.5	?/280 [272]	305	–	–	–	–	–	–	280	–	–	–	272

Table 2.3: Results of the Reinforced Coloring Tabu Search and results of the best performing coloring algorithms from the literature on all difficult graphs. The colorings of RCTS are publicly available at `http://www.info.univ-angers.fr/pub/porumbel/graphs/rcts/`

The algorithm acronyms correspond to the following papers: RCTS (Reinforced Coloring Tabu Search), ILS [Chiarandini and Stützle, 2002; Paquete and Stützle, 2002], VNS [Avanthay *et al.*, 2003], ALS [Devarenne *et al.*, 2006] , PCOL [Blöchliger and Zufferey, 2008], VSS [Hertz *et al.*, 2008], DCNS [Morgenstern, 1996], HGA [Fleurent and Ferland, 1996b], HEA [Galinier and Hao, 1999], AMCOL [Galinier *et al.*, 2008], MMT [Malaguti *et al.*, 2008], MCOL [Lü and Hao, 2010]. Notice that we indicate (in brackets) the results of the papers to appear in 2010, still in press at the defense of the thesis.

using problem-specific knowledge, could improve the results for other hard combinatorial optimisation problems, as already demonstrated for certain cases [Rodriguez-Tello *et al.*, 2008a; Rodriguez-Tello *et al.*, 2008b].

Chapter 3

Search Space "Cartography"

This chapter is devoted to a search space analysis, more exactly on investigating the spatial distribution of high-quality configurations. Such information will be later integrated in heuristic algorithms in order ot render them more "well-informed". The first objective was to design a local search algorithm capable of guiding itself toward unexplored regions (Chapter 4), but certain ideas are also employed by the evolutionary algorithm (Chapter 5) to ensure an appropriate population spacing.
Using a classical distance measure between colorings, we introduce the following *clustering* hypothesis: the high quality solutions are *not* randomly scattered in the search space, but rather grouped in clusters within spheres of specific diameter. We first provide intuitive evidence for this hypothesis by presenting a projection of a large set of local minima in the 3D space. More formally, we study the distribution of the distance between the high-quality configurations visited by a local search: the distance values are either very small (inter-cluster distances) or very large (intra-cluster distances). The chapter develops ideas from an article accepted in Computers and Operations Research [Porumbel *et al.*, 2010]; the overview of search space analyses is included an article presented at the LION conference [Porumbel *et al.*, 2009c].

Contents

3.1 Introduction

3.1.1 Analysing the search space

It is well known that the performance of all heuristic algorithms is strongly influenced by the search space structure. In fact, to design an effective heuristic one needs to exploit (explicitly or implicitly) some features of the search space [Streeter and Smith, 2006]. For illustration, several types of local minima structures can be exemplified for the satisfiability problem [Du and Pardalos, 2007, pp. 425–427]: isolated local optima, plateaus, valleys, bassins of attraction. Perhaps rather suprisingly at first glance, for many algorithms, the most difficult situation is not the classical local minimum, but the trap: the confinement of a group of local optima in a well. If trapped into such a structure, a classical local search would probably become locked looping between the local minima inside the well—even if it might be able to escape individual local minima.

There are many research threads related to search space analysis—see also [Streeter and Smith, 2006, §2] for a classification related to our interests. A research direction, especially in the context of evolutionary computing, consists of defining statistical measures to estimate the problem difficulty (i.e. convexity, ruggedness, smoothness or fitness distance correlation [Jones and Forrest, 1995]—see [Kallel *et al.*, 2001; Merz, 2004] for a summary of such measures and related issues. Other studies deal with the structural similarities between local optima (i.e. the "'backbone"' shared sub-structures) or with their spatial distribution—as we do in this chapter.

However, each problem has a different number of local optima and plateaux, each one with its own form, size or depth; furthermore, these local optima can be grouped following some patterns (in valleys, wells) and they can be concentrated in certain areas [Merz, 2004]. One can find numerous studies investigating the specific local optima properties for the most representative combinatorial optimization problems: boolean satisfiability [Zhang, 2004; Gerber *et al.*, 1998], graph coloring [Hertz *et al.*, 1994; Hamiez and Hao, 2004; Culberson and Gent, 2001], the traveling salesman problem [Stadler and Schnabl, 1992], the 0–1 knapsack problem [Ryan, 1995], graph bi-partitioning [Merz and Freisleben, 2000b], the quadratic assignment problem [Merz and Freisleben, 2000a], job shop or flow shop scheduling [Reeves and Yamada, 1998; Streeter and Smith, 2006], arc crossing minimization in graph drawing [Kuntz *et al.*, 2004],etc. It seems that the local optimum characteristics of the search space (the number of local optima, their space distribution, the topology of their basins of attraction) may indeed be very different from one problem to another, and even from one instance to another. However, all these studies conclude that the local optimum analysis has great potential to make a positive impact on the performance.

As also discussed in Section 1.1.2, it is more difficult to collect information "on the fly" (on-line learning) than to perform analyses in a pre-optimization stage. The inconvenience of any pre-optimization local optima analysis is obvious: it requires a-priori known information about the local optima of the objective function—and finding high-quality local optima is the actual goal of the main optimization stage. In fact, before the main optimization stage, very little advance information is usually available on the search space. A possible approach consists of using smaller instances so as to easily locate the optima to be analyzed [Hertz *et al.*, 1994]. It is important to notice that important insight can however be reached by analyzing artificial landscapes, e.g. the NK model [Kauffman and Levin, 1987; Tomassini *et al.*, 2008; Jones and Forrest, 1995], one-max or long k-path problems [Horn *et al.*, 1994].

In practice, the on-line integration of learned search space information in a search process remains a difficult problem. To achieve this, the search process needs to learn how to make better local decisions only using global information available at local levels. To overcome such difficulties in practice, the integration of a learning phase in the optimization process (“learning while optimizing”) seems very promising. Our approach, developing ideas of reactive search [Battiti *et al.*, 2008], aims at designing an algorithm capable of performing a self-oriented exploration.

3.2 The typical local search and the global vision

A very undesirable behavior of local search algorithms is to spend most of the time only in some specific parts of the search space. A particular classical case is to run into a local optimum and then to stay all the time only in its neighborhood. While the local optimum problem can be avoided by many thoroughly studied meta-heuristics, fewer efforts have been done to prevent the search process from looping between a limited number of basins of attraction and plateaus. Mmost local search algorithms move from configuration to configuration in the search space without recording too much data about the visited regions. Usually, at the moment of a given iteration, we have no information whether we are exploring a completely new region (i.e. the search has never visited any configuration close to the current one) or a previously-explored region.

Indeed, a local search algorithm is not commonly concerned with having a *global view* of its own the exploration process. In most combinatorial optimization problems, the algorithm does not take into consideration the relations between potential solutions (configurations) visited at *different* stages of the local search. Furthermore, most studies that analyze the search space structure (for example [Hertz *et al.*, 1994; Hamiez and Hao, 2004; Culberson and Gent, 2001] for graph coloring) focus more on providing theoretical information than on effectively using such information to improve an algorithm.

Given any local search process searching through a search space, some important questions might be:

- what does the exploration path look like ?
- which regions is the process more likely to explore?
- does the search process explore more than a few regions?

- what is the spatial distribution of high quality solutions?
- are they randomly scattered or are they grouped in structures?
- can the search process be guided toward a global optimum?

Clustering hypothesis In fact, the study in this chapter is devoted primarily to these issues. Using a search space distance metric (see Section 3.3.1.1 or detail description in Chapter 6), we introduce the notion of sphere: the set of colorings situated withing a certain distance (radius) from a (center) configuration. The clustering hypothesis is that the local optima are not randomly scattered in the space, but they form clusters of points that can be confined in *spheres* of specific diameter.

In this chapter, we consider the local search from Section 2.2, i.e. the basic Tabu Search algorithm without aspiration criterion but with reactive Tabu list. For readability issues, let briefly recall its features and construction. Essentially, TS iteratively moves from one coloring to another by modifying the color of a conflicting vertex until either a legal coloring is found, or a predefined stopping condition (i.e. the number of iteration) is reached. Each performed move (i.e. each new color assignment) is marked Tabu for a number of iterations, i.e. the Tabu list length T_ℓ. In this manner, TS cannot re-perform a move that was also performed during the last T_ℓ iterations.

This algorithm is indeed capable of avoiding certain local optima (and even small plateaus) by using the Tabu list. But, however, the above questions are still open and also essential. Is there any guarantee that TS is able to explore more regions in one week than in one hour? Unfortunately, as for many other local search algorithms, the answer is *no, it might explore roughly the same regions in one hour or in one week.* Experimentally speaking, many other papers using similar algorithms show that the results can not be improved by increasing the running time beyond a certain level (i.e. several hours for graph coloring). One can check that [Blöchliger and Zufferey, 2008; Hertz *et al.*, 2008] use running times of 1 hours and 10 hours and the improvement due to time is quite limited; we are skeptical that by using 100 or 1000 hours, such algorithms would reach new colorings.

3.3 Search Space Cartography

In this section, we explore the spatial distribution of specific configurations in the space search. Two different samples of configurations are considered: (i) the best optima $C_1^*, C_2^*, \ldots$ discovered by TS during independent runs, (ii) the best configurations visited by TS in one run. We consider each sample as a set of points in the search space Ω (a $|V|$-dimensional space), and we measure the distance between each two points with the following distance: the minimum number of neighborhood transitions to arrive from one point to another. The set theory provides the partition distance (see Chapter 6), that is very addapted to this situation.

First, we provide 3D visualizations for illustration: the points from the $|V|$-dimensional space Ω are mapped into the 3D Euclidean space such that a distance distortion is minimized. This is achieved with a classical Multidimensional Scaling (MDS) procedure. Then, we analyze the values of distances between all points and we provide evidence that they are grouped in clusters in the search space Ω. We make an estimate of the cluster diameter. Surprisingly, we observed that it does not closely depend on the graph type, but mainly on $|V|$. Our assumption is that many of these clusters can be confined in spheres of radius $\frac{1}{10}|V|$.

3.3.1 The operation of cartographing the search space

3.3.1.1 Search space distance: the minimum number of neighborhood transitions

Let us consider a combinatorial optimization problem with a search space Ω and a neighborhood N. To be able to perform correct visualizations of configuration positions, we need a distance measure reflecting the proximity according to neighborhood N—a step (N transition) should not “jump” over a long distance (and the minimum non-zero distance should be 1, corresponding to neighboring colorings). Let us consider the following distance measure: the minimal number of neighborhood transitions that one needs to apply on a configuration so that it becomes equal with the other. We call this a *neighborhood distance*, i.e. given two configurations $C, C' \in \Omega$, the distance between them is formally defined as the minimal number n for which there exist $C_0, C_1, \dots C_n \in \Omega$ such that: $C_0 = C, C_n = C'$ and $C_{i+1} \in N(C_i)$ for all $i \in [0 \dots n-1]$.

Fortunately for graph coloring, the set-theoretic partition distance fits well this definition; let us denote it by d. Using the partition representation (see Definition 1.2), the partition distance between coloring C and C' is the minimal number of vertices that need to be transferred from one class to another in C so that the resulting partition is equal to C'. Since transferring one vertex from one class to another is equivalent to changing a vertex color, the partition distance is indeed in accordance with the minimum number of neighborhood transitions from C to C'. Recall that, in this thesis, we deal with only one simple neighborhood function: the neighbors of a coloring are essentially obtained via color changes (see Section 2.2.1 for formal definitions).

For the partition distance, there exists a well-studied computation method using an $O(|V| + k^3)$ Hungarian algorithm—see the paragraph on “fast calculation procedure” in Section 4.2.2.1 for an overview. Complete details on this distance measure are given in Chapter 6, but for the moment it is enough to say that $d(C, C')$ needs to be a distance with a specific meaning: the shortest path of steps (neighborhood transitions, moves) between C to C' in the landscape defined by N. Similar cartographies can be realized with other distance measures, that can be defined for other problems using their specific neighborhoods (see examples in Section 4.5).

3.3.1.2 Multidimensional Scaling

Multidimensional Scaling (MDS) is a common procedure in data visualization for representing similarities or dissimilarities in data. It takes as input a matrix of distances (or dissimilarities between pairs of items) and maps them to a set of locations in the Euclidean space ($\mathbb{R}^2$ or $\mathbb{R}^3$) such that a loss function (i.e. *Kruskal stress* in our case) is minimized. In our implementation, the MDS procedure has three steps: data collection, data mapping, and model verification.

Step 1: The matrix of distances in the real search space Given a sample of k-colorings $\{C_1, C_2, \dots C_p\} \subset \Omega$, the procedure first constructs the matrix $D_{p\times p}$ where each element $D_{ij} = d(C_i, C_j)$ is the distance between C_i and C_j. We define the distance as the minimum number of color changes required to transform C_i in C_j, as discussed in the previous chapter.

Step 2: Generating $I\!R^3$ coordinates To obtain the corresponding locations in $I\!R^3$, we use the classical *cmdscale* algorithm for metric (classical) multidimensional scaling as implemented in the well-known R programming language for statistical data analysis[1]. The provided $I\!R^3$ locations are used to plot the 3D scatter graph and also to calculate the Euclidean distance matrix $d_{p\times p}$ between the points. More exactly, a function call of the form $cmdscale(D, k = 3)$ proved very efficient in providing a good set of 3D locations that were plotted with function `scatterplot3d`. (in fact, they can be even plotted with a different software); The distances in the 3D representation assess the spatial distribution of the real configurations in the Ω space.

Step 3: Quality assessment Since the isometry between the Euclidean distance matrix $(d_{p\times p})$ and the initial distance matrix $D_{p\times p}$ can not be exactly satisfied, the quality of the embedding is measured with a goodness-of-fit indicator. We measure the distortion with the classical stress proposed by Joseph Kruskal [Kruskal, 1964]:

$$s_{\text{fit}} = \sqrt{\frac{\sum_{1\le i,j\le p}(D_{ij} - d_{ij})^2}{\sum_{1\le i,j\le p} D_{ij}^2}}$$

The guideline provided by Kruskal in his seminal MDS paper [Kruskal, 1964] states that the representation is: *a*) poor if $s_{\text{fit}} > 0.2$, *b*) fair if $s_{\text{fit}} \le 0.1$, *c*) good if $s_{\text{fit}} \le 0.05$, *d*) excellent if $s_{\text{fit}} \le 0.025$ and *e*) perfect if $s_{\text{fit}} = 0$. In this chapter, even if the total number of points is very high, we present no 'poor' (i.e. $s_{\text{fit}} > 0.2$) representations.

3.3.2 Spatial distribution of the best configurations

Our first representation aims to provide a cartography (an intuitive map) of the locations of the best local optima in the search space. For the well-studied coloring DIMACS graph *dsjc*250.5 (quite small random graph with 250 vertices), the chromatic number is unknown but no algorithm ever found a legal coloring with less than 28 colors. We considered $k = 27$ and we tried to find the best k-colorings by performing hundreds of executions of the basic TS coloring algorithm; the best local optima we have ever found have 3 conflict. As such, assuming that a 3-conflict coloring is a global minimum, we performed 350 independent executions and we only collected the first global minimum discovered by each execution. Figure 3.1 (left) effectively plots the 350 best local optima *independently-discovered* by the 350 TS executions.

Figure 3.1 show some form of clustering of these "global optima" of this instances. One can place a few small spheres that cover all points—these spheres represent the regions

[1]`www.r-project.org`

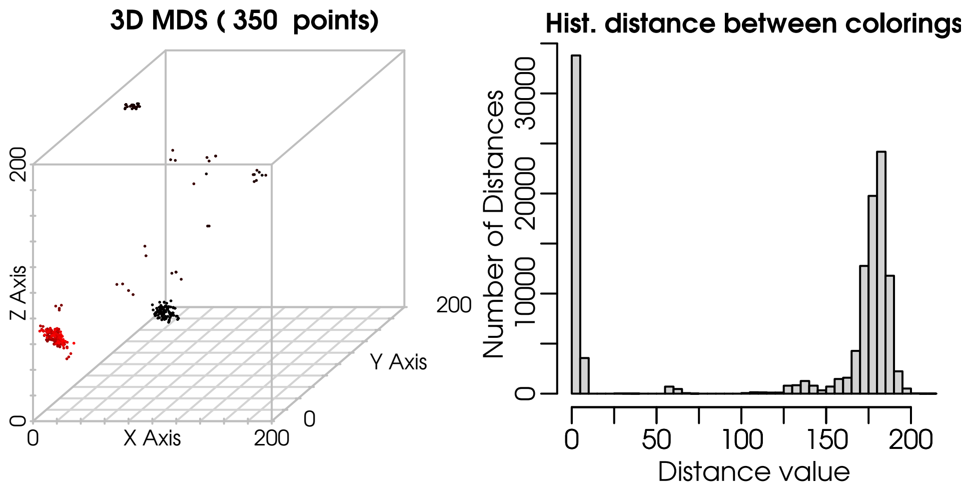

Figure 3.1: 350 global optima from the 250-dimensional search space of $dsjc250.5$ ($k = 27$) plotted in the 3D space (with stress $s_{\text{fit}} = 0.09$) by the MDS procedure (left) and the histogram of the real distance values between these colorings (right). Each coloring is independently found by a different run. The points are not randomly scattered, but grouped in clusters. The right figure confirms this: the distances between the points are either very small (inter-cluster distances) or very large (intra-cluster distances)

that usually attract the TS algorithm (e.g. very strong basins of attraction for TS). Figure 3.1 (right) shows the distribution of the distance values between any two global optimum (measured in Ω). Most values are either very small (less than $\frac{1}{10}|V| = 25$), or very large; the small distances correspond to inter-cluster distance while the large ones correspond to intra-cluster distances.

It is worth mentioning that this type of experiments can be performed only on *very rare configurations*, of excellent quality. Collecting widely distributed configurations would have been much easier, but any 3D projection would be less conclusive and less meaningful. Indeed, the solutions of the same graph ($dsjc250.5$) with $k = 28$ colors are distributed more uniformly, and so seem the solutions of other instances. This simple experiment required *several months of calculations* on a 2.7GHz processor, because it may take up to several days for a TS execution to find a point from this plot (i.e. a coloring with 3 conflicts with $k = 27$).

Finally, it might be interesting to note that the phenomena observed empirically in this section could have certain unexpected connections with similar findings from the statistical physics community. There is at least a graph coloring study (see [Zdeborová and Krzakała, 2007, Fig. 1]) describing how the global optima (legal colorings) are grouped in small clusters when the problem instance is very hard, i.e. at the COL/UNCOL (i.e. SAT/UNSAT) phase transition. On the other hand, if the instance is below the phase transition (and solutions can be easily reached), the global optima appear in a single giant cluster. However, the instance considered in this section is at the "COL/UNCOL" phase transition, and so, our clusters might correspond to the ones sketched in [Zdeborová and Krzakała, 2007, Fig. 1], see the fifth graph between c_r and c_s.

3.3.3 Spatial distribution of configurations sequentially visited in short time

In this section we examine the sequence of colorings visited by (quite) short TS processes, more exactly we investigate the arrangement of high-quality configurations:

Definition 3.1. *(High-quality configuration) We say that configuration $C \in \Omega$ is a high-quality configuration (i.e. it is deep, or hard–to–find) if and only if $f(C) \leq B_f$, where B_f is a fitness boundary (threshold). Otherwise, we say that C is a low-quality configuration.*

Given a problem instance (G, k) *and* an initial high-quality coloring C_0, we apply the TS algorithm starting from C_0. TS visits a series of neighboring colorings and let C_0 ,C_1, C_2, ... denote the high-quality configurations, those satisfying $f(C_i) \leq f(C_0)$—i.e. we consider the fitness boundary $B_f = f(C_0)$. In all our tests, the number of high-quality configurations represents only a very small fraction of the total number of colorings visited along the search; we ignore the colorings worse than C_0 because they are widely distributed and can be easily found. They are indeed scattered quite randomly in the space—i.e. even in the proximity of to the initial local optimum C_0, there should be numerous less fit colorings.

For example, for the problem $(le450.25c, 25)$: (i) it is hard to find a 0-conflict coloring[2], (ii) it is quite easy to find 1-conflict colorings if TS starts from an initial 1-conflict coloring, (iii) there are 2-conflict colorings scattered everywhere around the initial $1-$conflict coloring. For this graph and for a $1-$conflict starting coloring, we consider the other $1-$conflict configurations as relatively high-quality, and the 2-conflict configurations as relatively low-quality. If one also considers the low-quality configurations in this investigation, one would find points very uniformly distributed in the space.

We show in Figure 3.2 the MDS representations of the colorings resulting from this experiment. Two instances are considered: (a) random graph $dsjc1000.1$ starting from a 4-conflict coloring and (b) Leighton graph $le450.25c$ starting from a 1-conflict coloring. In order to limit the number of points (and the reliability of the MDS representation), we divide the TS exploration path in intervals of 100 and plot only the first coloring of each interval. More exactly, if the search visits the following high-quality configurations in this order: $C_0, C_1, C_2, \ldots$, we graphically depict only $C_0, C_{100}, C_{200}, \ldots$. The distances between the colorings visited at close moments of time (almost consequtively) are too small—even zero because TS makes many small loops. By taking only one coloring per interval, we can also cover a longer execution range with a limited number of points—an so, with acceptable stress (MDS distortion).

These 3D representations provide a good intuitive image of the exploration path. In the left graph, the exploration process starts from the front-bottom-left corner and passes from cluster to cluster until it reaches the right side; most colorings visited in-between do

[2]there are many state-of-the art algorithms that are unable to find legal 25-colorings.

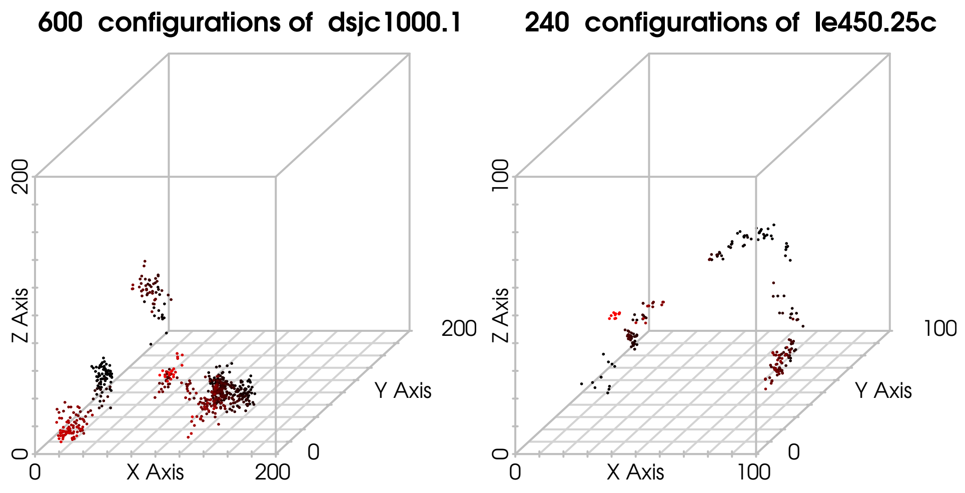

Figure 3.2: The high-quality colorings (with $f(C) \leq 4$) visited during 60000 iterations by TS for $G = dsjc1000.1$, $k = 20$ (left) and the high-quality colorings (with $f(C) \leq 1$) visited during 25000 iterations by TS for $G = le450.25.c$, $k = 25$ (right). The stress value is $s_{\text{fit}} = 0.19$ and $s_{\text{fit}} = 0.15$, respectively.

not appear in the graph because they have worse fitness values (low quality configurations). In the right graph, the exploration path is even more clear (it starts from the front-bottom-left corner and ends toward the front-bottom-right corner) but the clusters are closer—for $le450.25c$, the distances between clusters are between $15\%|V| = 67$ and $22\%|V| = 99$ (see also Figure 3.3, bottom right graph) because the Leighton graphs have a peculiar structure as we discuss in Section 4.4.4.

3.3.4 Spatial distribution of configurations visited in long runs

This section is devoted to a more formal (and less intuitive) analysis of much longer series of high-quality colorings visited by TS on *all* graph classes (Figure 3.3). We used a similar scenario as for Figure 3.2, but we examined much more configurations: indeed, *all* the high-quality colorings $C_0, C_1, \ldots, C_N$ (i.e. satisfying $f(C_i) < f(C_0)$ $\forall i \in [1..N]$, where $N = 40.000$) visited by TS are now considered for analysis (instead of the several hundreds of samples $C_{100}, C_{200}, C_{200}, \ldots$ considered in the previous representations). We compute all distances[3] $d(C_i, C_j)$ and, with a distance histogram, we show how many pairs (C_i, C_j) correspond to each distance value.

Figure 3.3 shows bimodal distance distributions, with either very small or very long distances between the C_i's: this confirms the existence of some well separated regions with high densities of C_i's (clusters). If we denote a "cluster diameter" by C_d, we observe that C_d varies from $7\%|V|$ to $10\%|V|$ depending on the graph, such that:

[3]Considering $N = 40000$ colorings, the number of distances to compute is not extremely large, we computed all $N \times N = 1.600.000.000$ distances in several hours.

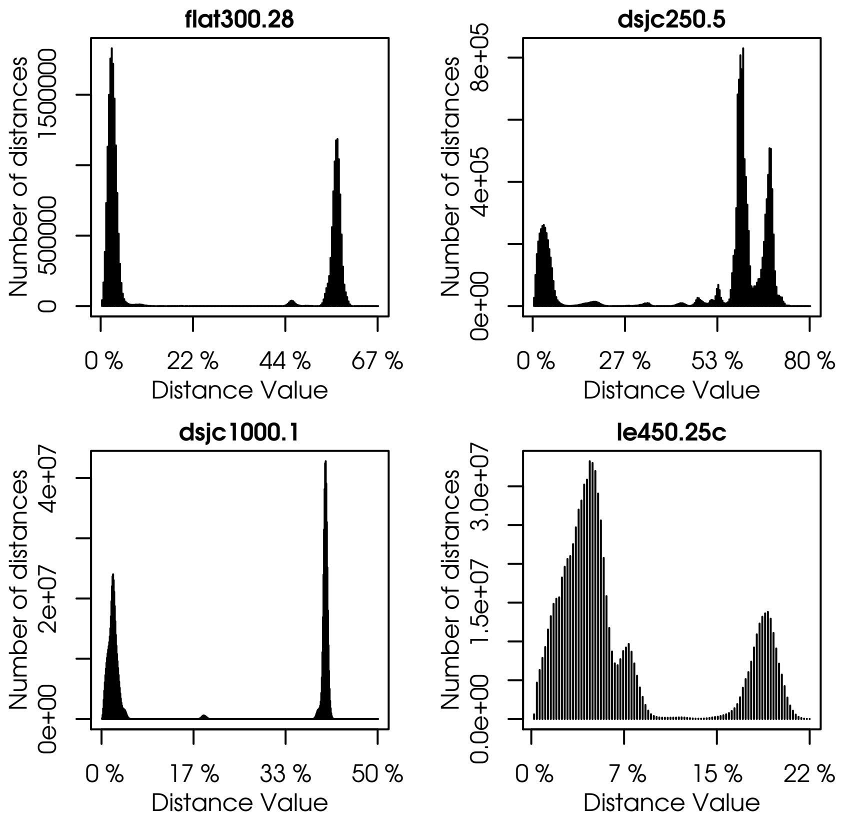

Figure 3.3: Histograms of the distances between each two of the $C_0, C_1, C_2, \ldots, C_{40000}$ (high-quality) configurations visited by TS; $f(C_i)$ is limited by: a) 4 conflicts for ($G = flat300.28$, $k = 30$), b) 3 for ($G = dsjc250.5$, $k = 28$), c) 4 for ($G = dsjc1000.1$, $k = 20$) and d) 1 for ($G = le450.25c$, $k = 25$).

- there are numerous pairs (i, j) such that $d(C_i, C_j) < C_d$;
- there are very few (less than 1%) pairs (i, j) such that $C_d < d(C_i, C_j) < 2C_d$;
- there are numerous occurrences of some larger distance values.

This statistical distribution of the C_i's mainly reflects the path of the search process through the search space—and not the arrangement of all existing high quality configurations. However, regarding the colorings visited by TS, this section confirms the *clustering hypothesis*, i.e. any two C_i's distanced by more than $\frac{1}{10}|V|$ (the largest possible value of C_d) belong to *different clusters* of high-quality configurations. We keep this estimation in the rest of the thesis and we use it to propose new search space exploration methods.

3.4 Conclusion of the chapter

A precise theoretical analysis of the cluster patterns is beyond the scope of this thesis; it would require algorithms with a complexity too large for integration into applications. However, our clustering hypothesis is very useful in practice: it is quite easy to exploit the fact that two high-quality configurations distanced by more than $\frac{1}{10}|V|$ belong to *different clusters*.

We will assume that the clustering hypothesis holds for any sequence of colorings that TS visits. The most important application of this principle is developed in Chapter 4 in which we present an algorithm for diversification, and one for intensification; both of them consider the search space organized in spheres with radius $\frac{1}{10}|V|$. Furthermore, since the TS procedure is also employed in the memetic algorithm from Chapter 5, we will use there the same principle, so as to keep a minimum spacing between population individuals, i.e. any two individuals of a population should be distanced by at least $\frac{1}{10}|V|$.

Chapter 4

Position Guided Local Search Algorithms Targeting Diversification and Intensification

We present two position-guided algorithms that work on top of a local search process (e.g. Tabu Search) so as to guide it toward certain targeted regions of the search space. For this, the search space is structured in spheres, and we take profit from the clustering hypothesis introduced in Chapter 3: the high quality configurations are *not* randomly scattered in the search space, but rather grouped in clusters within spheres of diameter $R = \frac{1}{10}|V|$. The first algorithm (TS-Div) uses a learning process so as to guide the underlying local search toward as-yet-unvisited *R-spheres*. The second algorithm (TS-Int) makes deep investigations in a "limited perimeter" around a given (promising) configuration. TS-Int employs a breath-first-search routine to enumerate all R-spheres from this "limited perimeter", and, each of these spheres is thoroughly explored by numerous independent TS processes. We experimentally observed that if such a "limited perimeter" contains a global optimum, TS-Int does not fail in eventually finding it. TS-Div ensures diversity, TS-Int enforces intensification, and together they reached very competitive results, in particular they colored for the first time the well-studied DIMACS instance $djsc1000.9$ with $k = 223$ colors; this chapter develops ideas from an article accepted in Computers & Operations Research [Porumbel *et al.*, 2010].

Contents

4.1 Introduction

4.1.1 Motivation and objectives

Compared to more refined evolutionary or swarm-based algorithms, the classical local search approach suffers from the fact that it does not exploit enough global mechanisms and features. Indeed, illustrations of global mechanisms can be found in many meta-heuristics: evolutionary algorithms use a distributed population along with recombinations or mutations to make “long jumps” in the search space, ant colony optimization introduced pheromone trails to induce interaction between globally distributed ants (agents), particle swarm optimization deals with self-organization, global feedback and stimergy to reach a “collective” behavior of swarms of agents, etc. This chapter is developed around a simple classical local search algorithm and tries to overcome the fact that it lacks any form of global vision or control—unlike the above meta-heuristics.

Indeed, without exploiting global information, all algorithm decisions are based only on microscopic local information (i.e. the neighborhood of the current configuration, and/or a short history), without any macroscopic global vision. Consequently, local search algorithms might not properly cover the search space, i.e. they can often visit the same regions over and over again, or they can simply get blocked in local optima. We show how to overcome this problem by integrating learned information in the exploration process. As such, the resulting local search algorithms from this chapter can effectively compete with more complex, more refined algorithms.

Let us now present the general ideas of the two algorithms. A central notion in both of them is the search space *sphere*: the set of configurations situated within a certain distance from a center configuration.

The first algorithm (TS-Div) is built on top of an underlying Tabu Search (TS), but also uses a learning component to interpret the exploration trace and to guide the TS process. This learning component keeps track of all visited configurations by recording all visited spheres (Section 4.2). To record a sphere, it is enough to record its center, and, even if the total size of the search space is huge, the number of spheres can be manageable.

This way, we experimentally proved that it is possible to record the exploration path by recoding a limited number of centers (see also Figure 1.1, p. 11). In order to guide the underlying search process toward as–yet–unexplored spheres, the learning component pays attention to keeping it at a certain distance from recorded spheres centers.

TS-Int (Section 4.3) is oriented to intensification. Given an input high quality configuration (provided by TS-Div, or by another algorithm), TS-Int performs in-depth explorations in a close perimeter around the input configuration. First, the sphere of the this configuration passes through a phase of "meticulous" investigation that launches numerous TS processes allowed to run inside this sphere. Each TS process might arrive at a sphere "exit point" that is recorded by TS-Int; new TS processes are launched until it is no longer possible to find distanced "exit points"—when we are sure of this, the sphere is considered "clear" (of global optima). Then, the sphere investigation is repeated with the spheres of the "exit points", i.e. each recorded "exit point" becomes the center of a new sphere to be checked later. Each newly-discovered sphere center is recorded in a ordered queue, and so, the centers are investigated in the order of their quality.

All experiments performed in this chapter use an underlying search process based on a version of the coloring Tabu Search algorithm from Section 2.2. While problem-specific knowledge can be essential for reaching good practical results, the main ideas of TS-Div and TS-Int relate only to the metaheuristic local search process. For a greater readability, let us present in Algorithm 4.1 the Tabu Search metaheuristic [Glover and Laguna, 1997]; the stopping condition is to solve the problem or to reach another stopping condition.

Algorithm 4.1: The basic Tabu Search (TS) metaheuristic

```
Input: Problem instance, (optional) the start configuration $C_{st}$
Return value: $f(C^*)$ (i.e. 0 if a solution is found)
$C$: the current configuration; $C^*$: the best configuration ever found
Begin
  1. $C$:=$C_{st}$ (choose $C$ = random configuration if $C_{st}$ is not specified.)
  2. while a stopping condition is not met
     (a) find the best $C' \in N(C)$ so that move $C \to C'$ is not Tabu
     (b) $C = C'$ (i.e. perform the chosen move)
     (c) if $(f(C) < f(C^*))$
           • $C^* = C$
     (d) mark $C$ Tabu for $T_\ell$ iterations
End
```

The most important details that need to be filled to solve an optimization problem with TS are the following: the objective function f, the neighborhood relation N and the Tabu list length. Regarding the coloring problem, these elements were already defined in Section 2.2 (see also Algorithm 2.1); we will employ the same algorithm used for the cartography in Chapter 3—i.e. the basic coloring Tabu Search from Section 2.2 with reactive Tabu list but without aspiration criterion. TS-Div and TS-Int use only one piece of additional problem-specific information: the sphere radius is set to $R = 10\%|V|$, following the clustering hypothesis from Chapter 3. In fact, this hypothesis is validated by the guided algorithms, as well as by the evolutionary algorithm in Section 5.5.4.

The interaction points between the guided (meta-)algorithms and Algorithm 4.1 are the following. The start configuration C_{st} mentioned in the input field of TS is used by TS-Int, i.e. TS-Int launches TS from a sphere center C_{st} and it stops the process when the distance from C to C_{st} is too large (reaching a “sphere exit point” becomes a stopping condition). TS-Div records certain configurations (sphere centers) visited by TS and it makes only one external modification on the skeleton from Algorithm 4.1: the Tabu list length can be artificially increased to force the search process leave the current sphere (the TS specification is reproduced inside Algorithm 4.2).

The remaining of the chapter is organized as follows. In Section 4.2 and Section 4.3, we present these two guided algorithms: TS-Div (assuring diversification) and, TS-Int (assuring intensification). Numerical results and discussions are presented in Section 4.4, followed by conclusions in the last section.

4.2 TS-Div—A position guided search to seek unexplored regions

The classical TS is useful to avoid re-visiting configurations visited in the near past (a very limited number), but it seems more difficult to generalize it to avoid all configurations visited in a distant past. This way, some potential solutions visited in the beginning of the execution can be revisited over and over again after some periods of time. There might be a limited number of local optima, with large bassins of attraction, that monopolize the execution time of a local search. While there are also many other well-studied methods to help a local search to escape a single basin of attraction, it seems more difficult to prevent it from *looping* between a limited number of basins of attraction. This is one of the main reasons for which, after a certain threshold, an increase of the execution time might not always improve the performance.

The TS-Div algorithm integrates a learning process for avoiding looping between already-visited regions. TS-Div employs an extended Tabu list length if it detects that it comes upon configurations that are too close, according to the distance function, to other previously-visited configurations, i.e. while it passes through an already-explored sphere. This strategy prevents the algorithm from revisiting such regions by the reinforced diversification phase associated with the extended Tabu list.

Figure 4.1 sketches the ideas behind TS-Div. While it is not possible to record the whole exploration path of a local search, one can record the spheres it visits. Indeed, the centers of all visited spheres are far less numerous, and so, they *can* be recorded by a learning component; this way, TS-Div can “realize” when the search process comes upon an already-visited sphere. In this situation, the Tabu list is increased so as to trigger diversification, i.e. to impeach the search process from revisiting the same sphere again. The difficulty comes from the fact that certain spheres contain some local optima that strongly attract any local search process passing nearby—the generic Tabu search from Algorithm 4.1 is always attracted to such points.

Generally speaking, one can see the recorded spheres as *Tabu spheres*, or at least, as “unrecommended spheres” that the search process is “discouraged” to visit. While the

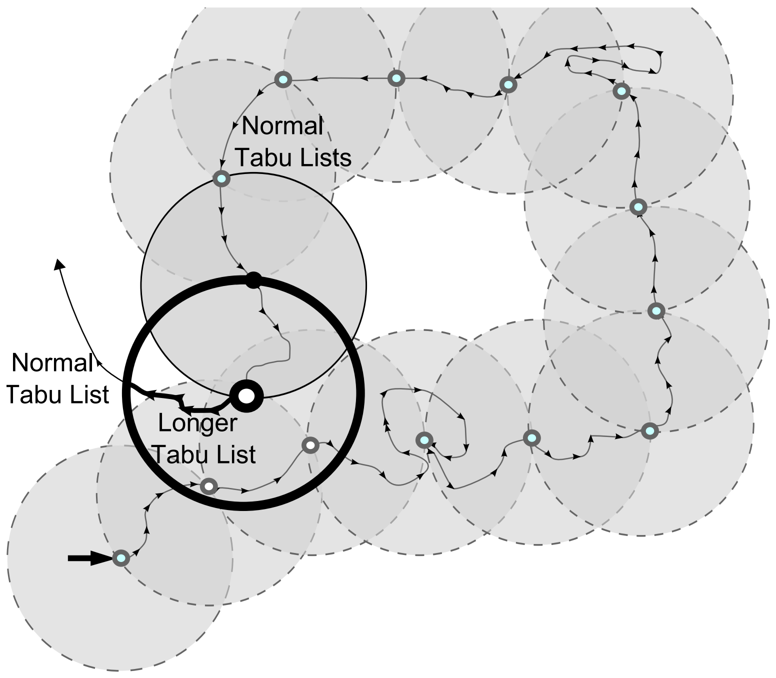

Figure 4.1: A schematic example of a TS-Div process exploring the search space. The curved continuous line represents the exploration path; to arrows indicate the entry and the exit point. The center of the thick sphere is contained in an already-visited sphere and TS-Div "realizes" that it needs diversification; in this sphere, a longer Tabu list is employed an so, the search is forced to diversify more, i.e. to find other spheres.

basic Tabu search can completely forbid certain configurations, it can do this only for a short time—i.e. it is not intended for imposing long-term Tabu statuses on configurations. This can be only achieved by performing the same operations at a higher level, i.e. by recording and forbidding spheres instead of configurations. In some sense, one can say TS-Div is no more no less than a *Higher Level Tabu Search*.

4.2.1 Formal TS-Div description

The TS-Div algorithm (see Algorithm 4.2) is based on two central processes: (i) the exploring process, using the TS procedure from Algorithm 4.1 (reproduced in Algorithm 4.2), (ii) the reactive learning process guiding the first one, Steps $4.(b) - (d)$ in Algorithm 4.2. The main tools of the learning process are the sphere, and the distance measure, defined as follows.

For the coloring problem, we use the partition distance from Section 4.2.2 below. Its calculation method is described in greater detail in Chapter 6, but for the moment it is enough to say that $d(C_a, C_b)$ represents a measure of type "shortest chain of neighborhood transitions" between C_a to C_b. It in accordance with the minimum number n for which there exist $C_0, C_1, \dots C_n \in \Omega$ such that: $C_0 = C_a, C_n = C_b$ and $C_{i+1} \in N(C_i)$ for all $i \in [0 \dots n-1]$. Distances of type "shortest chain of neighborhood transitions" can be found for other problems as well—see Section 4.5. Given such a distance, the search space R-sphere of C is defined as the closed sphere of radius R centered at C:

Definition 4.1. *(Sphere) Given a (center) configuration $C \in \Omega$ and a radius $R \in I\!N$, the R-sphere $\mathcal{S}(C)$ is the set of configurations $C' \in \Omega$ such that $d(C, C') \leq R$.*

Since we assume the clustering hypothesis (Section 3.3), we deal only with R-spheres of radius $R = \frac{1}{10}|V|$ in the rest of the chapter. In fact, the term "sphere" and "R-sphere" are used interchangeably. Two configurations C_a and C_b satisfying $d(C_a, C_b) \leq R = \frac{1}{10}|V|$ are called *close* or *related*; otherwise they are *R-distinct* or *R-distanced.* If $d(C_a, C_b) > \frac{|V|}{2}$, we say that C_a and C_b are *completely different.*

Taking as a basis the basic TS from 4.1, TS-Div performs the same operations but, in addition, it investigates the TS exploration path by recording the centers of all visited spheres. At a given iteration, we denote by C_p the last recorded sphere center and by C the current configuration. The first task of the learning component is to check the distance $d(C, C_p)$ to see whether C is still in the sphere $\mathcal{S}(C_p)$ of C_p—see step 4.(b) in Algorithm 4.2. While $C \in \mathcal{S}(C_p)$, the search process is still in the sphere of C_p and the algorithm iterates essentially as the basic TS does. We say that the search process is pivoting/revolving around pivot C_p.

As soon as the search process leaves the current sphere (i.e. $C \notin \mathcal{S}(C_p)$), the learning component concentrates on guiding decisions. It first compares C to the archive of all previously recorded configurations (procedure `Already-Visited` in Algorithm 4.2) to check whether it is entering in a previously-explored sphere or not. If C is not in the sphere of any recorded configuration, the learning component replaces C_p with C and records it in the archive, but the search process continues normally. Otherwise, if the search process is re-entering the sphere of a previously recorded configuration, the learning process intervene in the search process: a diversification phase is needed. For this purpose, we extend the default Tabu tenure T_ℓ with a T_{incr} factor, as explained in the next section.

Diversification using the Tabu tenure The Tabu list length (or Tabu tenure) provides a simple mechanism for controlling diversification and it has already been used in the literature—see also Section 2.4. Recall that the major effect of the Tabu list is that the algorithm can chose moves only among those that are not Tabu,i.e. the moves *not* performed during the last $T_\ell + T_{\text{incr}}$ iterations. As such, a longer Tabu list results in more diverse moves as the last $T_\ell + T_{\text{incr}}$ moves can not be repeated. For example, the algorithm cannot repeat the same series of moves unless the length of this series is greater than the Tabu tenure $T_\ell + T_{\text{incr}}$; any looping has at least $T_\ell + T_{\text{incr}}$ configurations. On the other hand, a shorter Tabu list is equivalent to stronger intensification; the algorithm returns much more easily to previously explored configurations re-performing moves completed in the near past.

To summarize, by varying the Tabu tenure (via T_{incr}), one controls the balance between diversification and intensification: the greater the T_{incr} value, the more diversification there

Algorithm 4.2: The TS-Div (meta) heuristic

```
PROCEDURE ALREADY-VISITED
Input: current configuration C
Return value: TRUE or FALSE
  1. Forall recorded configurations $C_{\text{rec}}$:
       • If $d(C, C_{\text{rec}}) \leq R$
           − Return TRUE
  2. Return FALSE

ALGORITHM TS-Div
Input: the search space $\Omega$
Return value: the best configuration $C_{\text{best}}$ ever visited
$C$: the current configuration
  1. Choose randomly an initial configuration $C \in \Omega$
  2. $C_p = C$ (the pivot, i.e. the last recorded sphere center)
  3. $T_{\text{incr}} = 0$ (the ``Tabu tenure'' increase triggered by TS-Div)
  4. While a stopping condition is not met
     (a) Set (next) $C$ = the best non-Tabu neighbor in $N(C)$
     (b) If $d(C, Cp) > R$
           • $C_p = C$
           • If ALREADY-VISITED($C_p$)
             − Then Increment $T_{\text{incr}}$
           • Else
             − $T_{\text{incr}} = 0$
             − Record $C_p$
     (c) Mark $C$ as Tabu for $T_\ell + T_{\text{incr}}$ iterations
                        /*$T_\ell$ =The internal default Tabu tenure from Alg. 4.1*/
     (d) If $(f(C) < f(C_p))$
           • Replace $C_p$ with $C$ in the archive
           • $C_p = C$ (i.e. ``recentering'' the current sphere)
     (e) If $(f(C) < f(C_{\text{best}}))$
           • $C_{\text{best}} = C$
  5. return $C_{\text{best}}$
```

is. A suitable control of T_{incr} guarantees that TS-Div keeps discovering new regions at all times; in fact, it can even guarantee that the process can never get stuck looping through already-visited spheres. The longer the time TS-Div spends only running into previously-visited spheres, the more it increments T_{incr}—and T_{incr} is only decremented (to 0) when the search process finds a new sphere. As such, the Tabu list is increased indefinitely until a sufficiently high value of $T_l + T_{\text{incr}}$ is guaranteed to break any looping between already-visited spheres (sooner or later). As such, TS-Div can discover new regions at all stages of the exploration, even in the very long run.

Note that there might be numerous artificial possibilities to create diversity when the algorithm realizes it is revisiting a sphere. For example, one could simply apply a random walk, or a perturbation commonly used in iterated local search algorithms. However, by applying artificial diversification operators, we consider the search process would risk to quickly reconstruct a configuration close to the avoided one.

4.2.2 TS-Div speed

The only problematic aspect of TS-Div concerns the slowdown introduced by the learning component that needs to compute numerous distances. Our objective is to keep the time spent by the learning component in the same order of magnitude as the time spent by the exploring component. To assure such a reasonable overhead, we focus on two crucial factors: the distance computation time and the number of distance calculations. If any of these two elements is not kept within reasonable limits, the speed of TS-Div can be compromised

4.2.2.1 Distance formal definition and fast calculation procedure

Using the partition representation (see Definition 1.2, page 15), the distance between coloring C_a and C_b is the minimal number of vertices that need to be transferred from one class to another in C_a so that the resulting partition is equal to C_b (equivalent to the minimal number of moves needed by TS to arrive from C_a to C_b). There exists a well-studied distance computation method using an $O(|V| + k^3)$ Hungarian algorithm—see [Gusfield, 2002] for a general set-theoretic approach or [Glass and Pruegel-Bennett, 2005] for the graph coloring application. However, in certain conditions (see Chapter 6), the distance can be determined in $O(|V|)$ time with a special method.

Basically, the distance is calculated with the formula $d(C_a, C_b) = |V| - s(C_a, C_b)$, where s is a measure of similarity defined as follows. Using the definitions from Section 1.2.2, $s(C_a, C_b)$ is $\max_{\sigma\in\Pi} \sum_{1\leq i\leq k} M_{i,\sigma(i)}$, where Π is the set of all bijections from $\{1, 2, \ldots k\}$ to $\{1, 2, \ldots k\}$ and M is a matrix with elements $M_{ij} = |C_a^i \cap C_b^j|$ [Gusfield, 2002; Glass and Pruegel-Bennett, 2005]. This similarity can be calculated by solving an assignment problem with the Hungarian algorithm. However, we applied the complete Hungarian algorithm only very rarely (less than 5% of all cases) because the calculation can be simplified by taking into account some problem particularities. Matrix M has at most $|V|$ non-zero elements and, as also stated in [Gusfield, 2002, §2], they can be filled in $O(|V|)$ with the appropriate data structure. Indeed, the non-zero elements of M are situated at positions $M_{C_a(x),C_b(x)}$ (with $x \in V$) and our algorithm works only with these elements.

Importantly, TS–Div does not require a precise distance value but we only have to decide if $d(C_a, C_b) > R$—i.e. $s(C_a, C_b) < |V| - R$. Since $s(C_a, C_b)$ is always less than $s'(C_a, C_b) = \sum_{1\leq i\leq k} \max_j M_{ij}$ (because $M_{i,\sigma(i)} \leq \max_j M_{ij}, \forall\sigma \in \Pi$), in many cases it was enough to check $s'(C_a, C_b) < |V| - R$ to decide $s(C_a, C_b) < |V| - R$. All maximums $\max_j M_{ij}$ can be identified by going through the non-zero elements of M, and so, $s'(C_a, C_b)$ can be computed in $O(|V|)$. More conditions in which the problem can be solved in $O(|V|)$ are available in Chapter 6.

As such, a distance computation requires (in average) approximately the same computation time as a TS iteration. The next section describes a method to keep the number of iterations and the number of distance computations in the same order of magnitude during long TS-Div executions. Thus, our distance computation procedure guarantees that the slowdown introduced by the learning component stays in acceptable limits.

4.2.2.2 Number of distance calculations

In terms of distance computations, the time needed to go through the archive (routine `Already-Visited`, Algorithm 4.2) is the most critical—at each call, TS-Div needs to check the distance from the current coloring to *all* colorings from the archive (see the `Forall` Loop in Step 1 of the `Already-Visited` procedure). Our objective is to keep the execution time of the learning component in the same order of magnitude as the exploring component. If the archive size exceeds a certain limit, the number of distance computations becomes too large, compromising the speed. However, the processing of the archive can become tractable if we focus the learning component on the high quality configurations, i.e. to allow the above routine to process *only* high-quality configurations, from the "deep" layers of the search space.

Following the analogy with the quest for the deepest location on the surface of the Earth (i.e. Mariana's Trench), one can also stratify our search space in deeper or upper layers—see [Du and Pardalos, 2007, p. 426] for more discussions on the multi-level search space model. In the upper layers, the danger of looping is smaller and the probability of finding a solution is very limited (the number of local optima itself is limited). As such, we can let the learning process concentrate only on the deeper search space layers without assuming too many risks. The only detail that needs to be filled is a formal threshold that defines our boundary between the deep search space and the upper search space, between high-quality and low-quality configurations. Recall (Definition 3.1, p. 45) that we say configuration $C \in \Omega$ is high-quality if and only if $f(C) \leq B_f$—B_f is the "formal" fitness boundary.

The fitness boundary B_f is automatically lowered and raised by TS-Div according to the balance between the number of computed distances and the number of iterations, so as to assure that the number of iterations stay in the same order of magnitude with the number of distance computations. This proved to be a good "thumb rule" for keeping the slowdown in acceptable limits. To be specific, B_f directly controls the learning overhead because the whole learning component (Step 4.(b)) is now executed only if $f(C) < B_f$. In practice, B_f varies from 5 conflicts to 20 conflicts; for some problems, even imposing $B_f = \infty$ could still result in an acceptable running speed.

Regarding the distance calculation from Step 4.(b), computing one distance per iteration does not pose too many problems—due to the small bound of the distance computation time. Furthermore, the distance computation from this step is not always needed: if $d(C_p, C) < R$, then TS-Div needs at least $R - d(C_p, C)$ moves to get out of the sphere of C_p. As such, after each distance calculation in this step, TS-Div can safely skip it for the next $R - d(C_p, C)$ iterations; we experimentally observed that more than 90% of the distance computations (from this step) can be skipped in this manner.

Furthermore, a number of simple optimizations details can significantly reduce the number of distance calculations along the search. Although we did not use this for graph coloring, one could also resort to increasing the radius R to reduce the archive size and the computational overhead. Another simple mechanism consists of transforming the archive into a queue that removes the oldest element at each insert operation. In this case, TS-Div becomes a memory-based *Higher Level Tabu Search* with two lists: (1) the traditional list of the last visited configurations that are forbidden, (2) the tabu list of spheres, used to avoid revisiting spheres visited in the recent past. The significance of the expression "recent past" would depend on the size of the queue which should be tailored according to the learning component overhead.

Experimental investigation on the number of computed distances This section only reinforces the fact that the number of distance calculation should stay in the same order of magnitude with the number of iterations. Since one cannot mathematically determine the exact number of distance computations of TS-Div (it depends on the search evolution), we studied it by performing an experimental statistical analysis on a classical DIMACS instance (G=$dsjc250.5$, $k = 27$). We run 10 TS-Div executions of 16 billion iterations an we investigate the memory and time required by any feasible value of B_f—in this experiment B_f is fixed.

$B_f(constant)$	0(equiv. TS)	5	6	7	**8**	9
Avg. nr. dist. calculations ($\times 10^6$)	–	1.5	31.7	877.7	**13743**	128932
Avg. time (hours:min)	120:56	120:51	120:37	122:32	**149:17**	312:43
Avg. nr. configurations in archive	–	523.8	6244.6	37810	**154462**	602523
Avg. memory (MegaBytes)	–	0.523	6.2	37	**154**	6025

Table 4.1: The resources consumed by the basic Tabu Search (column 2) and by TS-Div for several fixed values of B_f ($G = dsjc250.5$ and $k = 27$). All reported figures are averaged over 10 executions of 16×10^9 iterations each.

In table 4.1 we present (Row 2) the experimental count of all distance calculations and the number of hours spent in average on each execution (Row 3). The TS column shows that the basic TS would need about 120 hours to finish 16×10^9 iterations. If $B_f \leq 6$, the learning component overhead is negligible because the number of distance computations is very small (comparing to the number of iterations 16 billion). For $B_f \in \{7, 8\}$, the overhead is still acceptable (i.e. less than a quarter of the basic TS time) but it becomes too large for $B_f = 9$, as soon as the number of distance computations becomes too large comparing to the number of iterations (128 billion >> 16 billion). The memory required (Row 5) by the archive is proportional with the archive size (Row 4) because recording a configuration requires $|V| \times 4 \simeq 1000$ bytes[1], equivalent to approximately 1 kB.

4.3 TS-Int—A tree-traversal search of the problem space

A typical issue of classical local search algorithms can be described as follows: the search process arrives in a local minimum situated in the sphere of a solution, and it needs to choose the next moves. In the fortunate case, the search process chooses moves on the direction of the solution and, thus, the solution is soon discovered. But, there might exist

[1]Modern machines use 64-bit integers so an integer is stored in 4 bytes, but one can also use special 16-bit or 8-bit integers to save more memory.

Algorithm 4.3: Pseudocode TS-Int

```
Input: the starting configuration $C_s$
Return value: the best configuration ever visited
$Q = \{C_s\}$
While $Q$ is not empty
  1. $C_s$ =FRONT($Q$)
  2. Launch a TS process (Alg. 4.1) from $C_s$
     For each TS iteration, let $C$ be the current configuration and Do
     (a) If $d(C,C_s) > 2R$ then STOP TS and go to 3
     (b) If $d(C,C_s) > R$ and $C$ is high-quality then STOP TS and:
         i. If $d(C,C_i) > R$ forall $C_i \in Q$
            • PUSH($C$,$Q$)
  3. If $R$-sphere $\mathcal{S}(C_s)$ is ''clear''
     (a) POP($Q$)
```

millions of possible moves to escape from the local minimum; most of them might lead to different directions and the algorithm makes no distinction between them. Moreover, especially for TS-Div, if it chooses a move on a different direction and gets out of the sphere, it may never return again to it. To cope with this issue, TS–Int is introduced to methodically explore a limited perimeter, in the proximity of a given starting coloring C_s.

This limited perimeter is implicitely structured in spheres and TS-Int (see Algorithm 4.3) executes a number of *sphere investigation stages* (see Step 2). At each stage, a R-sphere $\mathcal{S}(C_s)$ is investigated by launching (from center C_s) numerous TS processes that explore only the proximity of C_s. Each such TS process is stopped when it reaches a high-quality configuration C outside $\mathcal{S}(C_s)$, an "exit point". If C verifies certain conditions (detailed below), it is recorded in a queue Q representing centers of spheres to be investigated later. After finishing a sphere investigation stage (TS-Int launched enough TS processes to consider the sphere is "clear"), TS-Int takes the next sphere center from Q (Step 1) and repeats the same sphere investigation stage from it (the old sphere center is removed from Q, see Step 3.(a)). Notice Q is sorted according to the objective function value so as to traverse the most promising spheres first.

A sphere investigation stage is finished only when $\mathcal{S}(C_s)$ can be considered "clear"—there is no global minimum in $\mathcal{S}(C_s)$ (that can be reached with TS). We explain next how TS-Int decides whether a sphere is "clear" or not. Recall that a TS process is stopped if it finds a configuration $C \notin \mathcal{S}(C_s)$ that is also high-quality (i.e. $f(C) < B_f$ as for TS-Div, see last paragraph below). Configuration C is inserted into Q only if Q does not already contain a sphere covering C (see step 1.(b).i in Algorithm 4.3). In this case, we say that the TS process was *successful*; otherwise, it was *unsuccessful*—it did not find new promising spheres. Notice we do not stop the TS process as soon as it exits $\mathcal{S}(C_s)$ because a search process can enter into and exit from a sphere several times before going away definitively from it. However, if the search process arrives too far from the center without finding any high quality solution (i.e. if $d(C,C_s) > 2R$, see Step 2.(a)), then search process is stopped and it is also labeled *unsuccessful*.

The conditions to declare $\mathcal{S}(C_s)$ as "clear" need to ensure that TS-Int keeps launching new TS processes from C_s as long as there are chances to discover new high-quality R-distanced exit points. We employ a condition that stipulate that TS-Int launches a first series of at least Y (usually $Y = 15$) executions from C_s and, in addition, it launches a number of X (we use $X = 10$) executions for every single *successful execution* (i.e. for each new insertion in Q). In this manner, a sphere investigation stage is finished only if the number of "unsuccessful" executions is at least $X = 10$ times higher than the number of successful executions—enough to consider the sphere has been explored from all "angles".

4.3.1 MDS representation of the TS-Int traversal

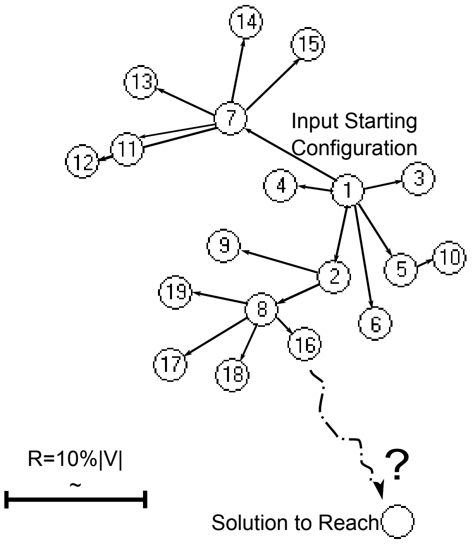

Figure 4.2: MDS representation of a TS-Int evolution for ($le450.25$,25). All points represent R-distanced 1-conflict colorings. They were discovered in the order of their numbers: 1 is the starting point, configurations $2, 3, \ldots, 7$ are discovered by the TS processes launched from point 1, etc. The distorsion level is acceptable (i.e. $s_{\text{fit}} = 0.05$).

We can say that TS–Int organizes the search space in spheres that are processed in a methodical tree-traversal manner—similar to a greedy breath-first-search or A*. The input start configuration is the root of the tree. Given any configuration C_s, an edge is added to link it to all new sphere centers discovered by TS processes started from C_s—recall that all sphere centers in Q are pairwise R-distanced. Similarly, each newly discovered sphere center C constitutes a new tree vertex linked to the configuration from which C was reached. The degree of C_s corresponds to the number of pairwise R-distanced new sphere centers reached from C_s. Basically, TS-Int launches TS processes from C_s as long as there are chances to discover new configurations, R-distanced from all other configurations in Q.

For illustration, Figure 4.2 plots a simplified TS-execution. Here, the solution, situated at distance $23\%|V|$ from the root vertex 1 (i.e. the input start configuration), is reached after an exploration of depth 3 (i.e. the tree on Figure 4.2 expanded on three levels). From experiments, the average degree of this exploration tree for this instance is actually 20. Consequently, the solution can be obtained by TS-Int after visiting at most $20^3 = 8000$ vertices. Indeed, we experimentally remarked that TS-Int always finds the solution in 10 runs out of 10 if it starts from a point within a distance of $\frac{1}{4}|V|$ from the solution (see more discussions in Section 4.4.3).

The algorithm proposed in this section (TS-Int) can be started from a single input coloring C_s but also from an archive of *completely different* configurations (i.e. $d(C_i, C_j) > 50\%|V|$ for all C_i and C_j in the archive). In fact, we can take the archive recorded by TS-Div, keep only a set of best, completely-different colorings and construct a queue for TS-Int. TS-Int is capable of searching in the proximity of the most promising points discovered by a different algorithm. Notice that TS-Int also uses the parameter B_f to make the distinction between high-quality and low-quality configurations. We took the value provided at the end of TS-Div, but, however it can also be automatically set (like TS-Div in Section 4.2.2.2), according to the number of distance calculations performed by TS-Int.

4.4 Results and discussions

In this section, we present detailed empirical results of the position-guided algorithms. While TS-Div is a classical “stand-alone” algorithm, TS-Int works in a post-optimization phase: it can be used to improve a certain input configuration (in our experiments, provided by TS-Div). All experimental tests are carried out on difficult problems from the well-established DIMACS Challenge Benchmark, See Section 1.2.3 (p. 16). We also discuss the influence of the graph structure on the *landscape surface* (associated with our neighborhood), as this can explain the behavior of other algorithms, too.

4.4.1 Experimental protocol

First, we need to point out that a TS-Div execution can be roughly equivalent to TS in the first iterations, while the archive is almost empty. The learning component intervenes in the exploration process only after a number of iterations, as soon as the exploration process starts to re-explore already-visited spheres. If the basic TS quickly solves a problem, TS-Div does not solve it more rapidly. The objective of TS-Div is visible in the long run, i.e. it only helps TS on the difficult instances where the basic TS fails.

To effectively test TS-Div, we perform 10 independent executions, each with a time limit of 50 hours. Within this time limit, TS-Div re-initializes its search with a random k-coloring each time it reaches 40 million iterations. Those restarts of TS-Div share the same archive of spheres. The statistics of the results are based on these 10 independent executions. Similarly, TS-Int is tested in the same experimental conditions, i.e. using 10 independent executions, each with a time limit of 50 hours. In Section 4.4.5, we will give some comments on the issue of running time with respect to the common practice of the literature.

Instance			TS-Div Algorithm		
Graph	χ, k^*	k	$\frac{successes}{executions}$	Iterations [$\times 10^6$]	Time [hours]
$dscj250.5$	?,28	28	10/10	4	< 1
$dsjc500.1$	?,12	12	10/10	42	< 1
$dsjc500.5$	?,48	48	2/10	7409	35
$dsjc500.9$	?,126	126	10/10	473	2
$dsjc1000.1$	?,20	20	2/10	2200	9
$dsjc1000.5$	?,83	87	5/10	2464	28
$dsjc1000.9$	?,224	224	8/10	1630	24
$flat300\_28\_0$	28,28	29	7/10	1186	8
$flat1000\_76\_0$	76,82	86	3/10	3020	33
$le450\_25c$	25,25	25	4/10	765	11
$le450\_25d$	25,25	25	2/10	1180	19
$r1000.1c$	?,98	98	10/10	47	< 1

Table 4.2: The results of TS-Div for a time limit of 50 hours. Columns 1, 2 and 3 denote the instance, the success rate (Column 4) is the number of successful executions out of 10; Column 5 and 6 show the average number of iterations and the average time needed to find a solution.

Regarding TS-Int, recall that it requires an input configuration C_s from which TS-Int starts searching for a solution (only a limited perimeter around C_s is explored). As such, the successfulness of TS-Int depends entirely on the distance from C_s to the solution. Typically, we first carry out a TS-Div execution, we collect the best colorings ever visited and provide them to TS-Int. These colorings are filtered so as to keep only completely different colorings as starting points for TS-Int; we finally present the results on the best input colorings.

4.4.2 Standard results of TS-Div and TS-Int

Table 4.2 reports the detailed results of TS-Div for several difficult instances (G, k)—especially those not easily solved by the basic TS. Columns 1–3 describe the instance, i.e. Column 1 is the graph, Column 2 shows χ (the chromatic number, or "?" if unknown) and k^* (the best known k when our article was accepted in 2009); Column 3 denotes the k for which we apply our algorithm. Columns 4–6 present the results of the algorithm, i.e. the success rate (the number of executions that solve the problem in 50 hours or less) in Column 4, the average number of iterations and the average time required to find a solution in Column 5 and 6, respectively. TS-Div improves the basic TS on most difficult instances because TS-Div never stops from exploring new regions—see also Section 2.5.4 for the performance of a very similar TS algorithm (f-RCTS uses an additional aspiration criterion but there are no major differences).

Table 4.3 presents the results of TS-Int on several instances for which TS-Div provides a starting configuration whose proximity is explored. Columns 1–3 have the same meaning as for Table 4.2, Column 4 shows the amplitude of the improvement (the number of conflicts of the start and the *end* configuration reported by TS-Int), Column 5 presents the success rate of achieving this improvement and Columns 6 and 7 denote the average computing effort (in iterations and in CPU hours, respectively).

Instance			Improvement	Success rate	Iterations $[\times 10^6]$	Time [hours]
G	χ, k^*	k	$f_{\text{start}} \longrightarrow f_{\text{end}}$	$\frac{successes}{runs}$		
*dsjc*1000.1	$?, 20$	20	$1 \to 0$	10/10	3774	12
*dsjc*1000.5	$?, 83$	86	$2 \to 0$	10/10	623	19
		85	$80^{*(k+1)} \to 0$	2/10	1453	39
*dsjc*1000.9	$?, 223$	223	$1 \to 0$	10/10	23	4
*le*450.25*c*	$25, 25$	25	$1 \to 0$	10/10	3410	10
*le*450.25*d*	$25, 25$	25	$1 \to 0$	10/10	6466	25
*flat*300.28	$28, 28$	28	$150^{*(k+2)} \to 0$	10/10	< 1	< 1
*flat*1000.76	$76, 83$	85	$74^{*(k+1)} \to 0$	10/10	1655	36

Table 4.3: Instances for which colorings are improved by TS-Int using a time limit of 50 hours. The input colorings are typically provided by TS-Div; however, the cells marked * indicate that TS-Int finds a legal k-coloring only starting from a legal $(k+1)$ or $(k+2)-$coloring.

TS-Int can reach a solution with 100% success rate (see Column 5 of Table 4.3) when it starts from an appropriate coloring (i.e. not too far from a solution, see also Section 4.4.3). Remarkably, it finds for the first time a legal coloring with 223 colors for the large graph *dsjc*1000.9. While TS-Div assures diversification, TS-Int is an intensification algorithm that can be systematically executed after TS-Div in order to (try to) improve its best colorings.

It is important to note that certain input start configurations provided to TS–Int (in Table 4.3) are easily accesible—i.e. except for *dsjc*1000.5, *dsjc*1000.9 and the rows marked *, all input configurations have been collected over a single TS–Div execution. For example, we solved ($dsjc1000.1, k = 20$) for the first time with TS-Int, not with TS-Div. The first TS-Div execution did not reach a solution but it found three completely different 1-conflict colorings; one of them leads TS–Int to the solution with 100% success rate. Similar conclusions can be drawn for the Leighton graphs in which we only applied TS–Int on the first 1–conflict configuration ever found by TS–Div.

4.4.3 TS-Int—finding the global optimum from an approximate location

TS-Int can be quite useful even in combination with other algorithms, especially when one could provide (by any means) an approximate location of the solution. In our main experimentation protocol, we consider that the configurations with lower conflict numbers have more chances to be close to a solution; thus, TS-Int always processes the spheres in the order of their conflict number. However, another possible "guess" of the solution location is obtained by considering that a legal k-coloring might be close to a legal $(k+1)$-coloring or even to a $(k+2)$-coloring.

This assumption worked perfectly well for the $flat300.28$ graph for which TS-Int finds a legal coloring with $\chi = 28$ colors starting from a legal coloring with $\chi + 2$ colors (the colors greater than χ are replaced with color 1). The graphs in this family are generated by adding edges only between the χ independent sets of an initial $\chi-$partition of V [Culberson and Luo, 1996]. A large proportion of the 30 classes of the legal 30-coloring are very close to some of the initial 28 independent sets, and so, TS-Int can easily reconstruct the rest of the coloring; the distance between the legal 30-coloring and the legal 28-coloring is only $7\%|V|$.

We experimentally observed that, if we provide a starting point within $\frac{1}{4}|V|$ distance (guess precision) from a solution, TS-Int finds the solution with a 100% success rate. These facts were observed on several graphs and for different initial colorings within $\frac{1}{4}|V|$ distance from a known solution (considering a time limit of 50 hours). Searching a solution within $\frac{1}{4}|V| = 25\%|V|$ distance (around the start configuration) typically requires a complete exploration of a tree on 3 levels because each edge corresponds to a distance of $10\%|V|$ in the search space—see also Figure 4.2 (p. 60). The number of levels that can be processed in a certain time limit depends on the exploration speed and on the average tree degree. Since TS-Int is also highly amenable to parallelization (at essentially 100% efficiency), one could speed up the algorithm by an order of magnitude by launching all processes from C_s in parallel. The required $\frac{1}{4}|V|$ precision of the solution location could thus be extended to $40\%|V|$ and even $\frac{|V|}{2}$ if sufficient computation time is allowed.

4.4.4 The influence of the graph structure on the landscape

We observed that, even if legal colorings are always very few and rare (for a difficult instance), the number of 1-conflict colorings (reached by our algorithms) can vary greatly. In the random graph case, before discovering a legal coloring, TS-Div usually visits between 3 and 20 colorings with one conflict. For the $flat300.28$ flat graph, TS-Div always directly descends to the legal 30-coloring from colorings with $4--5$ conflicts; as such, it can discover a solution even without visiting any 1-conflict coloring at all. At the other extreme, for the Leighton graphs, our algorithms can find thousands of 1-conflict colorings before finding a 0-conflict coloring (this is why it seems easier to solve $le450.25c$ by first finding a 1-conflict coloring with TS-Div and by applying TS-Int on it).

4.4.4.1 The coloring landscape for Leighton graphs

The explanation of these important contrasts lays in the structure of the graph: this structure defines the constraints between variables, and, implicitelly, the fitness landscape (as defined by our neighborhood). The Leighton graphs have a built-in 25-vertex clique [Leighton, 1979], and numerous 1-conflict colorings can share a common difficult conflict on this clique and be very different outside it. They form very large plateaus in which it is difficult to find the coloring that correctly colors the 25 vertices of the clique. This complements the information from the search space analysis in Chapter 3:

the distances between the clusters of the 1-conflict colorings (sequentially visited by TS on Leighton graphs, see the bottom right graph in Figure 3.3, p. 47) are significantly smaller than for a random graph.

4.4.4.2 The coloring landscape for flat graphs

The flat graphs are very different: they are constructed from a χ-partition of V such that a legal coloring has to assign a different color to each set from this initial partition. Any 1-conflict coloring has already identified most of these sets, and so, thet ransition to the solution is trivial. Indeed, the descent to the solution has always been very steep with any algorithm using *our* basic neighborhood (i.e. based on color changes, see Section 2.2). In fact, our algorithms can arrive at the solution even without ever visiting any 1-conflict coloring at all. We can say that the fitness landscape surface contains a unique almost vertical deep "well" (or hole) with the solution at its bottom. All intermediate quality colorings are situated in this "well"; we can say that a fitness-distance correlation is high.

This peculiar landscape structure can explain the behavior of several other algorithms on an instance like ($flat300.28$, $k = 28$). TS-Int is able to locate the "well" simply by using a legal 30-coloring as the starting point—this legal coloring implicitly identified many of the 28 initial independent sets. It is surely located in the proximity of the "well" and TS-Int can *not* miss the solution because this proximity is explored almost systematically. PartialCol [Blöchliger and Zufferey, 2008] can very easily find solutions with $k = 28$ colors simply because it uses a partition-based representation and neighborhood. VSS [Hertz *et al.*, 2008] can solve the instance with $k = 28$ because one of its search spaces is partition-based.

On the other side, even the best hybrid evolutionary algorithms fail on ($flat300.28$, $k = 28$) because they do not use a partition-based local search operator. Taking an easier instance ($flat300.28$, $k = 29$), we also observed that Evo–Div (the algorithm from Chapter 5) never finishes by reporting intermediate-value colorings, i.e. either it finds the solution, or it reports a best coloring with numerous (even more than 10) conflicts. Genetic algorithms perform much better on random graphs as they can combine color classes from completely different 1-conflict configurations to reach many new different good configurations (in other different areas). Regarding the $flat1000.76$ graph, all existing algorithms are very far from a $\chi = 76$ colors solution, and so, they rather behave like on random graphs (we observed that the solutions we have found with Evo–Div are very different—unlike those of $flat300.28$).

4.4.5 Running time of TS-Div and TS-Int

In our experimentations, TS-Div and TS-Int were allowed to run 50 hours per execution for a given coloring instance. We see that within this maximum time, TS-Div and TS-Int are able to reach very competitive results for the set of difficult graphs. Moreover, not all solutions required the maximum 50 hours computation time.

Let us mention that in the literature on graph coloring, it is a common practice to run a coloring algorithm several hours to several days to (try to) solve a hard coloring instance.

For example, some of the most recent coloring algorithms [Blöchliger and Zufferey, 2008; Hertz *et al.*, 2008] use running times of 10 hours for the largest instances. Now if we compare the number of iterations, the values required by TS-Div are even more comparable with respect to other local search algorithms. Indeed, the maximum iterations of TS-Div are in the order of billions (between 10^9 and 8×10^9) while the best local searches in the literature report the same order of magnitude (e.g. 2×10^9 iterations [Blöchliger and Zufferey, 2008, Table 6] or 3×10^9 [Hertz *et al.*, 2008, Table 5]).

Now let us insist on a more important point of TS-Div concerning the running time. One understands that, by its very nature, TS-Div will continually find new regions if it is given more computation time—it makes no important redundant exploration in the long run. Consequently, TS-Div will be able to find new or better solutions with the additional computation resources.

Notice that this is a desirable characteristic which is not verified by many existing algorithms. Very often, running them beyond some time (or iteration) threshold will not lead to better results simply because either the algorithms are trapped in deep local optima or because they re-explore again and again the same regions. TS-Div provides a simple, yet effective solution to this delicate issue because it is forced to discover new spheres at all stages of its execution. The same comment applies to TS-Int for which more computation time means more intensified exploitation of more spheres. Consequently, better results can naturally be expected.

4.5 Toward applications for other combinatorial problems

By carefully investigating the source code of the guided algorithms—see Algorithm 4.2 (p. 55) and Algorithm 4.3 (p. 59)—one observes that TS-Div and TS-Int do not exploit any feature specific only to the coloring problem. Indeed, the only components they require are:

- a search space (in our case, the set of all possible colorings)
- a neighborhood function (in our case defined by a color change)
- an objective function (in our case, the conflict number)
- and a search space distance (in our case, the partition distance).

Essentially, both algorithms are extensions of Tabu Search. The necessary condition to efficiently implement them for other problems is to find a search space distance measure : (i) with low computation complexity (comparing to a TS iteration) and (ii) that is in accordance with the neighborhood—TS should not “jump” over a long distance in one step. More formally, the distance should be in close relation with the minimum number of TS steps needed to go from one configuration to another. Like for any generic algorithm, the success of TS-Div or TS-Int are also dependent on problem-specific factors, i.e. a suitable choice of the radius R, or a method to define high-quality configurations.

However, as long as there exists such a distance measure with a reasonable computation time (that not significantly outweighs the computation time of a TS iteration), both algorithms can be applied to any combinatorial problem. TS-Int is not so sensitive to the distance calculation time and so, it can be more easily applied to other problems–it could be very effective when one has *any* other method to "conjectura" the location of the solution. However, with the distance measure, ideally, one should be able to group in a R-sphere all/most "equivalent" local optima—i.e. that share a common important "backbone" substructure. One can find several examples of easy–to–compute distances that can also be defined in this manner, by using some specific neighborhoods:

– the Hamming distance for problems with array representation using the 1-Flip neighborhood (i.e. constraint satisfaction problems with a neighborhood operator that consists of changing the value of a single variable),

– the Kendall tau distance [Kendall, 1938] for problems with permutation-based representation using a neighborhood defined by adjacent transpositions (i.e. the travelling salesman problem considering a neighborhood in which a move inverses two adjacent cities—the adjacent pairwise interchange neighborhood),

– the edit distance for problems with an array representation and with the neighborhood defined using edit operations.

Concerning the time needed to go through the whole archive, it can be substantially reduced (if necessary) in at least three ways: (i) by going only through the high-quality configurations (i.e. focussing on the "deep" layers of the search space), (ii) by increasing the value of the radius R and (iii) by transforming the archive into a queue that removes the oldest element at each insert operation. In the later case, TS-Div would behave like a Double-List Tabu Search, i.e. with two lists: (1) the traditional list of the recently visited configurations that are forbidden, (2) the new Tabu list of "recently" visited spheres, used to avoid revisiting spheres seen in the recent past. The significance of the expression "recent past" would depend on the size of the queue which should be tailored according to the overhead induced by learning component.

4.6 Chapter conclusions

Using empirical space search analysis that shows that the local minima of the coloring problem are grouped in clusters covered by spheres of $\frac{1}{10}|V|$ diameter (Chapter 3), we have devised a fast method to record the spheres of local minima visited by a local search process. The TS-Div algorithm records its exploration path (only by recording the spheres) and uses an additional learning process to discourage it from returning to already-explored spheres. Moreover, the TS-Div algorithm does not introduce any auxiliary user-provided parameters because B_f is automatically set.

The main objective of TS-Div is the global diversification of the search process: unlike the basic TS, TS-Div does not risk too many redundant explorations in the long run. As such, TS-Div is much more effective than the basic TS and it competes well even with the best algorithms from the literature. The search capacity of TS-Div is reinforced by TS-Int, which is an intensification-oriented algorithm used to better explore the proximity of the most promising configurations. It is able to systematically find a solution if the solution is situated within a certain distance from the starting configuration. TS-Int organizes the

Graphe	χ, k^*	TS–Div/Int	VSS	PCol	AmaCol	DCNS	HEA	MMT	MCol
		2009	2008	2008	2008	1993	1999	2008	2010
$dsjc500.1$	$?, 12$	*12*	12	12	12	12	–	12	12
$dsjc500.5$	$?, 48$	*48*	48	48	48	49	48	48	48
$dsjc500.9$	$?, 126$	*126*	126	126	126	126	–	127	126
$dsjc1000.1$	$?, 20$	*20*	20	20	20	21	20	20	20
$dsjc1000.5$	$?, 83$	85	87	88	84	88	83	83	83
$dsjc1000.9$	$?, 224$	***223***	224	225	224	226	224	225	223
$r1000.1c$	$?, 98$	*98*	–	98	–	98	–	98	98
$flat300.28$	$28, 32$	*28*	28	28	31	31	31	31	29
$flat1000.76$	$76, 82$	85	86	87	84	89	83	82	82
$le450.25c$	$25, 25$	*25*	26	25	26	25	26	25	25
$le450.25d$	$25, 25$	*25*	26	25	26	25	26	25	25

Table 4.4: Comparison between TS–Div/TS–Int and the best-known upper bounds at the moment when our article[Porumbel *et al.*, 2010] was accepted. All the reported colorings are available on the Internet for further research use: `www.info.univ-angers.fr/pub/porumbel/graphs/tsdivint/`

The algorithm acronyms correspond to the following papers: VSS [Hertz *et al.*, 2008], PCOL [Blöchliger and Zufferey, 2008], AmaCol [Galinier *et al.*, 2008], DCNS [Morgenstern, 1996], HEA [Galinier and Hao, 1999], MMT [Malaguti *et al.*, 2008], MCOL [Lü and Hao, 2010].

search space as a tree of connected spheres and applies a classical tree traversal algorithm to methodically investigate all spheres one by one.

To summarize, we devised a pair of algorithms that assures both the diversification and the intensification tasks by guiding underlying search processes. Table 4.4 presents a comparison between the best results obtained in this study and the best results from the literature. Five papers in this table are very recent (year 2008) as we took into account only the best algorithms; moreover, some of these cited papers present in fact more than one algorithm version—but Table 4.4 shows the best k reported by any of them. The population-based heuristics (last four columns) are traditionally the most effective and, on some instances, they all find better results than all local searches. However, our method is also very effective comparing with all state-of-the-art algorithms and it even finds for the first time a new legal coloring for a very hard DIMACS instance ($dsjc1000.9$).

Chapter 5

Diversity Control and Grouping Recombination in the Evolutionary Approach

This chapter is devoted to evolutionary algorithms, introducing both problem-specific contributions and also certain general ideas of evolutionary computing. Population diversity is ensured by a generally applicable procedure to control spacing between individuals. Using a search space distance measure, this procedure decides what individuals are acceptable in the population, what individuals need to be replaced and when to apply mutations. We present a very "well-informed" multi-parent crossover: it uses several features of the parent color classes (i.e. class conflicts, class size, vertex degrees) to select those used as building blocks for offspring construction. In fact, the algorithm is well-informed in several senses. For example, the high-quality coloring clustering information from Chapter 3 enables us to define the threshold of minimum population spacing. In fact, there are several connections to previous chapters (e.g. the new evaluation function from Section 2.3.1 is employed in the local search), but we will briefly recall the basic ideas. Most of the best-known upper bounds are reached (often with 100% success rate) and an upper bound is even improved. The chapter is based on a 12 page article selected among the three best-paper nominees at Evocop 2009 [Porumbel *et al.*, 2009a]; the complete work is submitted to a leading journal.

Contents

5.1 Introduction

As indicated in the introduction of this study (see Section 1.2.1), an important class of coloring heuristic algorithms is represented by evolutionary hybrid methods. Most of these methods are basically memetic algorithms—evolutionary algorithms incorporating local search—and they constitute one of the most effective approach for our problem (Section 1.2.4).

However, a major drawback known in all fields of evolutionary computing is the risk of premature convergence—and this seems an an essential problem for graph coloring, too. To deal with this risk, we use the distance measure between individuals and we show how to use it to ensure a healthy *spacing* between individuals. We present an generally-applicable population strategy to impose a certain distance between individuals in memetic algorithms. The new strategy can guarantee that premature convergence is always avoided, while not sacrificing population quality. At the same, this procedure also reduces another important risk specific for small populations, i.e the risk of failing to adequately cover the search space [Reeves, 1997].

Another challenging issue in evolutionary computation is to design a crossover that is meaningful for the problem, i.e. that preserves good features of the parents and disrupts the bad ones [Radcliffe, 1994]. For graph coloring, it is very useful to consider colorings as vertex set partitions [Galinier and Hao, 1999]—recall a partition of set V is a splitting of V into non-overlapping classes (or groups). Indeed, most recombination schemes select certain classes from two parent partitions and use them as building blocks to assemble the offspring [Galinier and Hao, 1999; Malaguti *et al.*, 2008; Hamiez and Hao, 2001; Malaguti and Toth, 2008]. In fact, this is in accordance with the principles of grouping genetic algorithms in which promising groups (genes) need to be transferred from parents to offspring by inheritance [Falkenauer, 1998]. An essential issue is to correctly choose the *best* classes (or groups) to be used for offspring construction. While previous approaches consider only the class size or the class conflicts, we introduce a refined measure to rank the parent classes and, implicitly, to better choose the information that is passed to offspring.

We also investigate which specific features can render an individual very fit (high quality). Such information is very useful along the chapter, i.e. to design a well-informed evaluation function in the local search, to determine the appropriate number of parents in the recombination, etc. By also studying the arrangement of very fit individuals, we define an appropriate border between "close" and "distant" individuals. An important principle of our diversity strategy is to work only with distant individuals. The resulting algorithm, which will be called Evo–Div, matches most of the best-known upper bounds and also finds a new solution for a well-studied graph. Finally, we clearly evaluate the contribution of each new major component or technique.

Next section recalls the basic notions and presents the general design of the coloring evolutionary algorithm. The new multi-parent crossover is detailed in Section 5.3. The method of controlling the population diversity/spacing is discussed in Section 5.4. Section 5.5 is devoted experimental results and we finish by presenting final remarks and conclusions in the last section of the chapter.

5.2 Generic graph coloring memetic algorithm

5.2.1 Specific terminology in evolutionary computing

In order to present our algorithm, let us notice that we will employ a slightly different terminology, according to the well-established tradition of the evolutionary computation community. Comparing to the rest of the thesis, the main differences are the following:

individuals They represent the potential solutions or configurations employed in the local search part. As such, we say that all individuals form a *population* $POP = \{I_1, I_2, \ldots I_{|POP|}\}$;

fitness function In accordance with terminology from the Darwin's theory, we say that evolutionary algorithms try to find the most fit individual, the one with the lowest value of the fitness function. In our case, the fitness function is the objective function f, equivalent to the number of conflicts—see Definition 1.3, page 15;

new evaluation function $\widetilde{f_{\text{eval}}}$ Our hybrid algorithm uses a local search procedure that employs a new evaluation function. While in Section 2.3 we introduced *two* new evaluation functions $\widetilde{f_1}$ and $\widetilde{f_2}$, in this chapter we only use $\widetilde{f_{\text{eval}}} = \widetilde{f_1}$.

We only use this new evaluation function in the local search procedure of the memetic algorithm; in the rest of the chapter, only the classical fitness function is used.

5.2.2 The memetic algorithm

The skeleton of Evo–Div (see Algorithm 5.1) uses basic ideas from previous similar algorithms [Fleurent and Ferland, 1996a; Galinier and Hao, 1999; Costa *et al.*, 1995;

Malaguti *et al.*, 2008; Dorne and Hao, 1998a; Galinier *et al.*, 2008; Morgenstern, 1996], but it also bring several new features: routines to control population spacing, new mutation operators, or the possibility to use $n \geq 2$ parents. Most coloring evolutionary algorithms are *memetic* because recombination alone can not produce individuals of sufficiently high quality and local improvement is needed. Indeed, it is difficult to perfectly blend color classes taken from parents, and so, the crossover of high quality parents can result in less fit offspring. The goal of the local search is to fix this blending and also to intensify the search around the coloring constructed by the crossover operator.

Our memetic template in Algorithm 5.1 uses function `acceptOffspring` to detect what offspring solutions do not fit certain *spacing* criteria. The mutation is triggered only when the natural reproduction process can no longer produce sufficiently spaced offspring through, i.e. more exactly, when it reaches a number of failed tries (i.e. `maxRejections`). As in the natural evolution, mutations arise very rarely. The *stopping condition* is either to find a legal coloring or to reach a predefined time limit. In our experiments, the local search procedure is the most time consuming operator.

Algorithm 5.1: Evo–Div: Evolutionary Hybrid Algorithm with Diversity Strategy

```
Input:  the search space $\Omega$
Result:  the best configuration ever reached
 1. Initialize (randomly) parent population $Pop = (I_1, I_2, \ldots, I_{|Pop|})$
 2. While a stopping condition is not met
    A. rejections$= 0$
    B. Repeat
        1. $(I_1, I_2, \ldots, I_n)$=RandomParents($Pop, n$) /* $n \geq 2$ */
        2. $O$ =Crossover($I_1, I_2, \ldots I_n$)
        3. If rejections$\geq$maxRejections
          a. $O$ =Mutation($O$)
        4. $O$ =LocalSearch($O$,maxIter)
        5. rejections++
     Until AcceptOffspring($Pop, O$)
    C. $I_R$ = ReplacedIndiv($Pop$)
    D. $Pop = Pop - I_R + O$
```

The parent selection (`RandomParents`) is quite classical and it simply consists in choosing n different individuals uniformly at random from the population. In this manner, the "survival of the fittest" selection pressure is only at the replacement stage, which thus becomes essential. To complete the algorithm description, the rest of the chapter presents the black-box components from this template, most notably: the crossover operator (routine `Crossover`, see Section 5.3), the spacing strategy (routines `AcceptOffspring` and `ReplacedIndiv` and `Mutation`, see Section 5.4). The local search improvement procedure (`LocalSearch`) is independent of the evolutionary scheme and it is described briefely in the next section below.

5.2.2.1 The Local Search Procedure

The `LocalSearch` routine implements a version of the Tabu Search algorithm discussed in Section 2.2. We briefly recall its features and construction. Essentially, this algorithm iteratively moves from one coloring to another by modifying the color of a conflicting vertex until either a legal coloring is found, or a predefined number of iterations (i.e. $maxIter = 100000$) is reached. Each performed move (i.e. each new color assignment) is marked Tabu for a number of iterations, i.e. the Tabu list length T_ℓ. In this manner, Tabucol cannot re-perform a move that was also performed during the last T_ℓ iterations. Importantly, our procedure finally returns a *random configuration from all* visited colorings with the same minimum conflict number.

Recall (Chapter 2) that we set the Tabu list length using $T_\ell = \alpha \cdot f(C) + random(A) + \left\lfloor \frac{M_{\text{cst}}}{M_{\max}} \right\rfloor$, where α, A and $M_{\max}$ are predefined parameters, M_{cst} is number of the last consecutive moves that kept the fitness function constant (Section 2.4), and $random(A)$ return a random integer in $[1..A]$. Concerning the values of these parameters, we use: $\alpha = 0.6$, $A = 10$ (as previously published in [Galinier and Hao, 1999]), and $M_{\max} = 1000$. The last term is a new reactive component only introduced to increment T_ℓ after each series of $M_{\max}$ iterations with no fitness variation. This situation typically appears when the search process is completely blocked looping on a plateau; a longer Tabu list can more easily trigger the search process diversification that is needed in this case (see also Section 2.4).

Well-informed evaluation function At each iteration, the Tabu search algorithm has to choose a non-Tabu move that leads to a coloring with the minimum number of conflicts. If there are several possible choices, the traditional approach is to take any *at random*. We discussed in Section 2.3.1 that it is useful to introduce additional criteria to differentiate between colorings with the same conflict number. In this chapter, we use the evaluation function $\widetilde{f_1}$, and since there is no risk of confusion with $\widetilde{f_2}$, we denote $\widetilde{f_{\text{eval}}} = \widetilde{f_1}$. Formally, using (2.5) from Section 2.3.3.2, we recall the evaluation function:

$$\widetilde{f_{\text{eval}}}(I) = \sum_{\{i,j\} \in CE(I)} \left(1 - \frac{1}{2|E|\delta_i} - \frac{1}{2|E|\delta_j} \right),$$

where δ_i denotes the degree of vertex i (that is non-zero, recall that G is connected) and $CE(I)$ is the set of conflicting edges (see also Section 1.2.2). Using this function, the local search continues to choose moves leading to the best conflict number, but the random choice is based on this function, i.e. the lower the $\widetilde{f_{\text{eval}}}$ value of a coloring, the more chances it has to be chosen as the next coloring.

The basic principle behind this evaluation function will also be used in other components presented later in this chapter—e.g. the recombination operator employs degree information to better distinguish color classes. Chapter 2, as well as Section 5.5.4, show that such degree-based differentiation is useful for certain graphs with a very high degree variation. Essentially, we exploit the idea that it is more difficult to reduce the conflict of an edge with end vertices have higher degrees. If the algorithm has to choose between two 1-conflict colorings, it prefers the one with a more isolated edge in conflict, i.e. with the two conflicting vertices of lower degree, involving less constraints.

5.3 Well-informed crossover

A coloring can be seen as a vertex set partition (Definition 1.2), and this appears to be the best interpretation to design an effective crossover. In this context, the recombination consists in choosing k color classes from the parents and in assembling them for constructing the offspring. The goal is to make the offspring inherit the best k color classes, or the most fit, the ones that bring the highest contribution on quality. For example, if one can select k conflict-free classes (independent sets) that cover V, then one can use them to construct a legal coloring. This is usually *not* possible, and so, an essential question needs to be addressed in this section: *how* to select the *best k* color classes from parents. Furthermore, while previous studies typically consider two parents, we propose a generalized framework that determines the appropriate number of parents for each instance.

In order to establish a meaningful class ranking, our first concern is to design a well-informed scoring (fitness) function for classes. For this, we propose 3 scoring criteria: (i) the number of conflicts inside the class, (ii) the number of vertices in the class and, (iii) the sum of the degrees of the class vertices. The first criterion is very important in terms of quality and it assumes that a color class with fewer edges in conflict is always preferable to a class with more conflicts. However, one often encounters classes with no conflicts as Evo–Div converges to near-optimal configurations. A second criterion is needed: if one has to choose between two classes with the same number of conflicts, one prefers the largest one. Finally, there are instances for which the color class sizes are quite homogeneous and a third criterion is used for further discrimination: the sum of the degrees of the class vertices. The idea behind this third criterion is that a vertex of lower degree is more isolated and easier to color (see also Section 5.2.2.1); as such, a class with higher degree vertices is more valuable.

Formally, the crossover operator, hereafter called Well-Informed Partition Crossover (WIPX), is specified in Algorithm 5.2. It first searches (Steps 2.A and 2.B) in all parents for the class with the best (minimal) score. After assigning it to the offspring (Step 2.C), it chooses the next most fit class and repeats. At each step, all class score values are calculated by ignoring the vertices that already received a color in the offspring (see Step 2.A.1). WIPX stops when k colors classes are assigned; any remaining unassigned vertex receives the color k (Step 3).

An important risk of this crossover is to inherit most classes only from one parent, especially if there is a (very fit) parent whose classes “eclipse” the others. However, the similarity between the offspring and the parents is implicitly checked afterward by the diversity control procedure that rejects the offspring if it is too similar to any existing individual.

Algorithm 5.2: The Well Informed Partition Crossover WIPX

```
Input: parents $I_1, I_2, \ldots, I_n$
Result: offspring $O$
1. $O$ =empty, i.e. start with no vertex color assigned
2. For $currentColor = 1$ To $k$
    A. Foreach parent $I_i \in \{I_1, I_2, \ldots, I_n\}$
        Foreach color class $I_i^j$ in $I_i$
          1. Remove from $I_i^j$ all vertices already assigned in $O$
          2. $conflicts = |\{(v_1, v_2) \in I_i^j \times I_i^j : (v_1, v_2) \in E\}|$
          3. $classSize = |I_i^j|$
          4. $degreeCls = \sum_{v \in I_i^j} \delta_v$          /*$\delta_v$=degree of $v$*/
          5. $score[I_i^j] = conflicts - \frac{1}{|V|}(classSize + \frac{degreeCls}{|E| \times |V|})$
    B. Set $(i^*, j^*) = \arg\min_{(i,j)} score[I_i^j]$
    C. Foreach $v \in I_{i^*}^{j^*}$
        $O[v] = currentColor$
3. Foreach unassigned $v \in O$
    $O[v] = k$
```

5.3.1 The best inheritable features and the number of parents

An important principle in crossover design for grouping problems is to promote promising groups (genes) and to avoid as much as possible to disrupt good features existing in parents [Falkenauer, 1998]. Our goal is then to identify what "good features " are and how to promote them in the inheritance process. The quality of very fit colorings can reside in individual classes (genes) of high quality (e.g. independent sets), but also in their blending, or mixture. We can say that the quality can be due to excellent individual genes but also to high epistasis—e.g. productive interaction between genes. Importantly, the number of parents determines the fragmentation of the class blending from each parent. Since each parent provides $\frac{k}{n}$ classes (in average) to the offspring solution, a higher n results in fewer classes selected from *each parent*—equivalent to more disruptions of the existing class blending (from each parent). Our goal comes to deciding what is more important: preserving the class blending or maximizing the probability of inheriting good individual classes?

Certain instances have solutions with numerous very *small classes*—e.g. 4 vertices (in average) for an instance with $|V| = 1000$ and $k = 223$. It is not difficult to find independent sets of this size, but the difficulty consists of mixing many small classes in an optimum way. This situation typically appears for dense graphs, for which one needs a large number of colors k and the average class size (i.e. $\frac{|V|}{k}$) becomes very low. For small classes, it is better to use two parents so as to preserve the existing blending of color classes.

At the other extreme, there are the sparse graphs for which one needs fewer colors and *large class sizes*, i.e. 50 vertices (in average) for instances with $|V| = 1000$ and $k = 20$. In this example, the quality of very fit colorings resides rather in large independent sets, than in their blending. Consequently, it is recommended to use more parents because: (i) this does not disrupt high-quality features (independent sets), and (ii) a larger number of input classes for recombination (e.g. $n \times k$ with $n = 4$) increases the probability of selecting and inheriting very good individual classes (for the previous case with $k = 223$, $4 \times k \approx 1000$ input classes would have been too much, making the result too chaotic). The generic rule is to use two parents for instances with small classes, three for instances with average classes, or four for very large classes. To be specific, our rule is to use:

1. $n = 2$ if $\frac{|V|}{k} < 5$
2. $n = 4$ if $\frac{|V|}{k} > 15$
3. $n = 3$ otherwise.

5.3.2 Related work

The algorithm 5.2 provides a general framework for designing recombination in coloring problems, and, more generally, in partitioning problems. In fact, by modifying the class scoring (step 2.A.5), one can generate other crossovers. For example, to obtain the Greedy Partition Crossover from [Galinier and Hao, 1999], one basically needs to score each class with the class size (i.e. *score* = -*classSize* in step 2.A.5) and to set $n = 2$. The first three crossovers from [Malaguti and Toth, 2008] can also be replicated in this manner. The recombination from [Hamiez and Hao, 2001] uses a different framework and it is used to construct offspring only from independent sets. A similar version can be obtained by setting *score* to ∞ if the class has conflicts, or to *classSize* otherwise. It seems that the idea of constructing offspring only with independent sets has a positive influence on certain particular graphs; see also the discussion on more crossovers versions in Section 5.5.4.

In all these methods, after inserting the k classes in the offspring, some problematic vertices may remain unassigned. In previous studies, a greedy procedure is used to assign a color to each of them, so as to minimize the conflicts over the graph. However, we consider this operation also induces a risk in disturbing good color classes—because it could add the new conflicts following quite superficial greedy decisions. We prefer to assign to all these problematic vertices the same color k (Step 3), and so, only the last color class is disturbed with new conflicts. The task of assigning better colors to these problematic vertices is thus left to the subsequent local search steps.

One should be aware that in the large graph literature coloring literature, there have been also tried other types of crossover operators, based on the color-oriented (array) encodings [Fleurent and Ferland, 1996b; Malaguti and Toth, 2008], on unifying pairs of conflict-free sub-classes [Dorne and Hao, 1998a], on sexual reproduction using a splitting of the graph into two parts [Marino *et al.*, 1999], distance-preserving crossovers [Tagawa *et al.*, 1999], etc.

5.4 Maintaining and creating diversity

Our diversity strategy has two objectives. The first one is to maintain an appropriate spacing between individuals at all times of the execution. This ensures that the population individuals can not all converge toward the same point of the search space—i.e. premature convergence is avoided and diversity is preserved. However, this can not guarantee that the algorithm is able to *create* diversity, i.e. to permanently discover new search space areas. For this reason, our second objective is to make the population distribution *evolve* along time, so as to continually move from old regions to new ones. In this manner, even if the population of our memetic algorithm is small, it can cover numerous regions over the time.

There are many different means to measure diversity[REFS], but in this study we use an indicator based on distances between individuals. Our diversity is thus based on the spacing indicator S, defined as the average distance between two individuals. In addition, the *minimum spacing* $S_{\min}$ refers to the minimum distance in the population. The two objectives of our spacing strategy can be formally expressed as follows:

- keep $S_{\min}$ above a specific target minimum spacing R
- make the average spacing S as high as possible.

Basically, the first point is addressed by the offspring rejection procedure and the second by the replacement operator. They correspond to routines `AcceptOffspring` and `ReplacedIndiv` in Algorithm 5.1 and they are presented in Section 5.4.2 and Section 5.4.3 below.

5.4.1 Search space distance metric

We brieefly recall the distance metric on which the whole spacing strategy is based. Using the partition coloring representation (see Definition 1.2, Section 1.2.2), the partition distance (call it d) is the minimum number of elements that need to be moved between classes of the first partition I_A so that it becomes equal to the second partition I_B. Importantly, the distance between I_A and I_B is equivalent with the minimum number of local search steps (color changes) required to arrive from I_A to I_B. This way, our local search procedure can not jump over a long distance in one step.

A distance calculation procedure is sketched in Section 4.2.2.1 and complete details are provided in Chapter 6. However, the speed of this calculation procedure is not essential for the evolutionary algorithm. We did not observe any significant slow-down caused by distance computations. The local search operator is much more time consuming as it is quite long (we use $maxIter = 100000$ iterations). The distance represent a number of vertices (whose colors need to be changed), and so, it can only take values between 0 and $|V|$; this is why we usually report distance values in terms of percentages of $|V|$.

5.4.2 Offspring rejection

Since the first objective of the diversity strategy is to maintain a target minimum spacing (denoted by R), we insert an offspring in the population only if its distance to each existing individual is greater than R. Consequently, if an offspring solution O is situated at a distance of less than R from an individual I, the rejection procedure detects this issue (routine `AcceptOffspring` returns False in Algorithm 5.1) and performs one of the following actions:

1. rejects O, **if** $f(O) > f(I)$—i.e. if O is worse than I;
2. directly replaces I with O, **if** $f(O) \leq f(I)$.

However, in both cases, we consider that O is not satisfactory enough because it does not bring diversity to the population. As such, Evo–Div does not pass to the next generation and repeats the whole reproduction process, starting with the parent selection, see the `Repeat-Until` loop in step 2.B. of Algorithm 5.1. The reproduction process is repeated until the offspring solution is distanced enough from the existing individuals, see also Section 5.4.2.2.

5.4.2.1 Target minimum spacing R

A delicate issue in this procedure is to determine a suitable value of the minimum space (denote by R, referring to the sphere radius). Recall (Definition 4.1) that we denote by $\mathcal{S}_R(I)$ the closed sphere of radius R centered at I, i.e. the set of individuals $I' \in \Omega$ such that $d(I, I') \leq R$. If I is a high quality individual, an appropriate value of R should imply that all other high quality individuals from $\mathcal{S}_R(I)$ share important color classes with I, i.e. they would bring no new information into the population (or they are structurally related to I). We have to determine the maximum value of R such that all high quality individuals, that are structurally *unrelated* to I, are situated outside $\mathcal{S}_R(I)$.

Since all individuals in the population are local minima obtained with Tabu Search, we will use the value of R determined by the analysis of the TS exploration path. Recall (Chapter 3) this classical scenario: start from an initial local minima I_0, and let Tabu Search visit a sequence of high quality individuals I_0, I_1 ,$I_2, \ldots I_N$ (i.e. we count only the individuals satisfying $f(I_i) \leq f(I_0)$, $\forall i \in [1..N]$). After recording all these individuals up to $N = 40000$, we computed the distance for each pair (I_i, I_j) with $1 \leq i, j \leq N$ and we constructed a histogram to examine the number of occurrences of each distance value.

This histogram (Chapter 3.3.4) showes that the distribution of the distance value is bimodal, with numerous occurrences of small values (around $5\%|V|$) and of some much larger values. This provides evidence that the $I_i's$ are arranged in distant groups of close points (clusters); the large distances correspond to inter-cluster distances and the small ones to intra-cluster distances. If we denote a "cluster diameter" by C_d, we can say that C_d varies from $7\%|V|$ to $10\%|V|$ depending on the graph: To determine a suitable R value, it is enough to note that any two individuals situated at a distance of more than $10\%|V|$ (approximately the highest possible C_d value) are not in the same cluster—because (ideally) they have certain different essential color classes. In conformity with

Section 3.3.4, we set the value of R to $10\%|V|$ for all subsequent runs. We say that two individuals distanced by less than $R = 10\%|V|$ are "too close"; otherwise they are "distant", or R-distanced.

5.4.2.2 Reactive dispersion via mutations and increased R

Ideally, the target minimum spacing R can be assured only by the natural reproduction process,i.e. through crossover and local search. One of the principles of our spacing policy is to ensure diversity without sacrificing quality, and so, Evo–Div applies "artificial" mutations as rarely as possible—i.e. only as a last-resort tool. The mutation only consists of perturbing a certain number (*mutation strength*) of randomly chosen vertices. To be specific, all colors of these vertices are first erased and new colors are sequentially assigned using a greedy criterion (minimize the generated conflicts). The strength is initially R (i.e. R vertices are perturbed), but, if the offspring solution resulting after local search is still rejected, the strength is doubled at the next `Mutation` call (in the `Repeat-Until` loop of Algorithm 5.1). The mutation strength can be gradually increased (i.e. R, $2R$, $3R\dots$) until a sufficiently-high strength (at most $|V|$) enables the local search to produce a distanced offspring solution.

Increasing R to disperse the population Recall (Algorithm 5.1) that the mutation is triggered only ofter *maxRejections* tries that failed to produce offspring solutions distanced-enough from existing individuals. By using a very high value of *maxRejections*, Evo–Div ensures a very low overall number of mutations throughout the search (see Section 5.5.1.1). However, mutations can become more frequent in certain special situations that require enforcing more diversity, i.e. when the search process is blocked looping on particular search space structures. Let us now show *how* to make an evolutionary search "realize" when there is not enough diversification.

For illustration, the population can converge toward a very stable state in which: (i) the average spacing is less than $2R$ and (ii) all individuals have the same fitness (the best-ever so far). In this situation, a large part of the population can be confined in a sphere of radius $2R$ that contains many R-distanced local optima—this can be due to some particular search space structures, i.e. numerous plateaus confined in a deep "well". To deal with this issue, we propose a reactive dispersion mechanism so as to trigger numerous mutations in the subsequent generations to help the population to leave this problematic $2R$-sphere. This mechanism also resort to the minimum spacing R that is also essential in the diversification/intensification balance. By doubling its value in this situation, most subsequent offspring solutions from the $2R$-sphere will be rejected. By also reducing considerably *maxRejection*, the dispersion mechanism allows much fewer tries of natural offspring birth, ensuring more frequent mutations and more diversification.

The same reactive dispersion is also applied to unblock the search on the following situation: when too many offspring solutions are continually rejected since the beginning of the search—e.g. when the average of "rejections per generation" becomes three times more than the normal (of about 0–1.25, see Section 5.5.1.1). This indicates that the population is distributed around certain local optima with very strong basins of attractions, so that the natural reproduction process usually leads to offspring in the same basins. By using the reactive dispersion mechanism, mutations are triggered more rapidly and most part of the population soon leaves the current positions. Defining "too many rejected offspring" is not difficult if we take into account that the average number of rejected offspring per generation is between 0 and 3. Experiments show that if it reaches 5 (for the whole execution, since the beginning), the diversity-enforcing strategy can be applied.

Note that, while mutation is an essential operator in evolutionary strategies or real-encoded genetic algorithms, it is seems less useful in combinatorial optimization, or, at least, for graph coloring. Indeed, we are not aware of previous graph coloring algorithms presenting a mutation operator with a very positive effect on performance. While a mutation operator can easily assure diversity, this does not typically compensate the associated quality loss. This is the main reason why mutations should be used sparingly.

5.4.3 Replacement strategy

At each *generation*, an existing individual is chosen by the `ReplacedIndiv` routine in Algorithm 5.1 to release a slot for the offspring solution. While the reproduction process is important for discovering new promising areas, this replacement operator is also very important because it decides what areas are abandoned. The relative position of the offspring solution is not taken into account in this decision. Recall that the offspring rejection procedure checks that the offspring solution is not too close to any individual, i.e. the offspring has to be the representative of a different distant area. As such, the distribution of the population permanently moves from certain already-visited areas to new areas.

5.4.3.1 Direct replacement

However, population replacement is not carried out only with the `ReplacedIndiv` routine. Until the reproduction process generates an offspring solution sufficiently different from existing individuals, we consider that the population distribution is stagnant and Evo–Div does not pass to the next generation. As detailed in Section 5.4.2, if there is an existing individual I close to the offspring O, then I can be directly replaced by O (if $f(O) \leq f(I)$). With this move, the population distribution does *not* actually evolve toward new regions, but this is only a "side" move performed so as to intensify the search in the R-sphere that contains O and I. Since this R-sphere contains two high-quality individuals reached independently, we consider the R-sphere is promising and it deserves intensification.

This direct replacement can also result in violating the constraints of the target minimum spacing, e.g. if there are I_1 and I_2 such that $d(I_1, I_2) > R$, $d(O, I_1) < R$, and $d(O, I_1) < R$, directly replacing I_1 with O would lead to a minimum spacing of $d(O, I_2) < R$. However, this is an anomaly that is solved at the next call of `ReplacedIndiv`. Indeed, this routine first finds the closest individuals I_a and I_b, and, if $d(I_a, I_b) < R$, the less fit of them is eliminated. In this manner, out of the three initial *close* colorings O, I_1 and I_2, only one will eventually survive, assuring that a population slot is always set free for individuals exploring more regions. Importantly, population stagnation is avoided.

5.4.3.2 Standard replacement

In standard cases $d(I_a, I_b) > R\ \forall I_a, I_b \in Pop$ and the `ReplacedIndiv` routine is not typically concerned with assuring the target minimum diversity. Its objective is to increase the average spacing and, for this, it needs to get rid of small distances between existing individuals. In addition, it should also respect the ”survival of the fittest” principle. Since the parent selection is uniformly random, the replacement stage is essential for both spacing and quality.

Generally speaking, the standard elimination procedure (see Algorithm 5.3 below) selects two very close individuals that candidate for elimination and only the least/less fit of them is eliminated. The first candidate C_1 is chosen by a random function using certain fitness-based guidelines (via the `AcceptCandidate` function). The second candidate C_2 is chosen by introducing the following *spacing criterion*: C_2 is the closest individual to C_1 respecting the same fitness-based guidelines as C_1.

Algorithm 5.3: The Replacement (Elimination) Function

```
Input:  population $Pop = (I_1, I_2, \ldots, I_{|Pop|})$
Result:  the individual to be eliminated
1. Repeat
     $C_1$ = RandomIndividual($Pop$)
   Until AcceptCandidate($C_1$) (fitness-based acceptance)
2. minDist = maximum possible integer
3. Foreach $I \in Pop - \{C_1\}$
     If $d(I, C_1)$ <minDist
        If AcceptCandidate($I$)
           • minDist = $d(I, C_1)$
           • $C_2$ = $I$
4. If $f(C_1) < f(C_2)$
     Return $C_2$
   Else
     Return $C_1$
```

The `AcceptCandidate` function separates the first half of the population from the second half—with respect to the *median* fitness value; additionally, the best individuals are also treated separately. As such, this function always accepts a candidate C_i for elimination if C_i belongs to the second half, but it accepts C_i only with 50% probability if C_i belongs to the first half. Only the best individual is fully protected; it can never become a candidate for elimination—unless there are too many best individuals (more than half of the population) in which case *any* individual can be eliminated. As such, the role of the first half of the population is to permanently keep a sample of the best individuals ever discovered. The first half of the population stays quite stable in comparison with the second half that is a diversity-oriented sub-population, changing very rapidly. This might recall the principles of scatter search, a population-based heuristic using an intensification set and a diversification set—see [Hamiez and Hao, 2001] for a graph coloring scatter search algorithm.

5.4.4 Related research and ideas

Principles of diversity preservation strategy can be found in several research threads from the literature. In diversity-guided or diversity-controlling genetic algorithms [Ursem, 2002; Shimodaira, 1997; Zhu, 2003], one uses an indicator of overall population diversity so as to choose the genetic operators and their application probability. For example, [Zhu, 2003] uses a diversity indicator based on the average hamming distance to adapt the mutation and crossover rates. In [Ursem, 2002], one employs intensification operators (selection and recombination) when a diversity indicator is high, or diversification operators (mutation) when the diversity is too low. In this research thread, one does not really need a distance measure between individuals, but only an overall population diversity indicator. For example, [Ursem, 2002] employs an indicator of statistical dispersion around the average value on each variable, making it particularly useful in continuous optimization. Generally speaking, the population diversity can be measured by numerous indicators, depending on the field (see the refs in [Zhu, 2003, p. 1]).

A distance-based study is the MA—PM algorithm [Sörensen and Sevaux, 2006], in which the population management also uses an offspring rejection procedure. In MA—PM, if an offspring solution does not satisfy the diversity criterion, it is immediately mutated. In contrast, we prefer to repeat the natural reproduction process until it brings diversity to the population. This way, we do not sacrifice "quality for diversity"; in addition, we also bring forward a completely new replacement operator and we set the target minimum spacing (i.e. the "diversity parameter") according to motivations of clustering. For these reasons, Evo–Div can also compete with the best heuristics on the underlying problem.

We also mention that, in memetic algorithms, it is a common strategy to give attention to the fact that the offspring solution needs to be different enough from its parents. Since local search is used, there is a high risk that the recombination of two very fit and close parents leads to similar solutions. To avoid this, a good idea is to always mate *distant* parents and, in addition, one can also take care that to generate the offspring solutions at equal distances from each parent. For example, the crossover in [Galinier and Hao, 1999] takes care to inherit half of the color classes from one parent, and half from the other. Similar techniques can be found for other combinatorial problems, e.g. see the distance preserving crossover for the traveling salesman problem [Freisleben and Merz, 1996]. However, our crossover is not intended to ensure equal distances between the offspring solution and parents; this issue is indirectly addressed only by the rejection and replacement strategies.

More distantly related, the crowding distance introduced by Deb et. al [Deb *et al.*, 2002] is used for solution ranking in multi-objective numerical optimization. An essential difference comparing to this method is that the crowding distance is measured in the objective function space—i.e. it relies on differences between fitness values of the individuals [Deb *et al.*, 2002; Coello *et al.*, 2004; Rajagopalan *et al.*, 2008].

5.4.4.1 Other evolutionary computation fields

In multi-modal continuous optimization, one needs to locate all the global optima of a function with several "peaks". In this context, niching methods (e.g. fitness sharing or crowding [Mahfoud, 1995a; Mahfoud, 1995b; Smith *et al.*, 1993; Deb and Goldberg, 1989; Miller and Shaw, 1996; Goldberg and Richardson, 1987; Beasley *et al.*, 1993; De Jong, 1975; Cedeño and Vemuri, 1999; Smith *et al.*, 1993]) were introduced for the "formation and maintenance of *stable subpopulations*" [Mahfoud, 1995a; Mahfoud, 1995b]. Each subpopulation is devoted to a peak of the multi-modal function and it can be seen as a sub-species [Miller and Shaw, 1996] that exploits a "niche".

Compared to niching, Evo–Div does not encourage the population to specialize permanently on several niches, but it insists on repeatedly moving from region to region. For multimodal functions, the niches correspond to the several peaks of the function and one often promotes crossover only inside the subpopulations [Miller and Shaw, 1996; Cedeño and Vemuri, 1999] (intra-niche crossover). In our memetic algorithm, we prefer "inter-niche" crossover because this is the best tool to discover new regions. Our goal is to make the distribution of a *small* population evolve very rapidly so as to explore millions of different niches along the time, i.e. to maintain *and* create diversity throughout the search.

Fitness sharing [Goldberg and Richardson, 1987] is a very popular niching technique that is based on the following principle: if two individuals are distanced by less than a cut-off distance (the niche radius or the sharing parameter σ_{share}), their fitness value is reciprocally penalized [Goldberg and Richardson, 1987; Miller and Shaw, 1996; Smith *et al.*, 1993; Deb and Goldberg, 1989; Beasley *et al.*, 1993]. Although this approach is different from ours, an essential part of it consists in defining a suitable "niche radius", a concept resembling our minimum spacing R. While our strategy it to reject inserting offspring on a radius of R around any existing individual, by sharing, one would only penalize such a close offspring solution. In both cases, if an inappropriate radius is used, the efficiency is significantly decreased. We set this radius using clustering motivations; sharing theories on niche radius are available (see [Beasley *et al.*, 1993, §5.1]) and they often rely on specific assumptions on the multimodal continuous function (e.g. that it has an a-priori known number of evenly distributed peaks).

Crowding selection (e.g. de Jong's crowding [De Jong, 1975], Mahfoud's deterministic crowding [Mahfoud, 1995a], and others [Cedeño and Vemuri, 1999]) tries to "'induce niches by forcing new individuals to replace individuals that are similar genomically"' [Smith *et al.*, 1993]. In this manner, a new individual replaces one from its own niche or subpopulation and it minimizes changes on the parent population, i.e. it preserves the diversity of the existing mixture [Miller and Shaw, 1996]. This resembles our direct replacing procedure, but, while our operation also induces a form of niching, stable niches/clusters are not actually recommended for large discrete spaces. Our standard replacement chooses the eliminated individual regardless of the offspring's position and, it encourages the population to continually move from "niches" to "niches".

Final Remarks In light of the above, the idea of explicit population control using distance metrics, seems, to a certain extent, overlooked in single-objective combinatorial optimization. However, there are several reasons for which it can be very useful to improve memetic algorithms with small populations—algorithms that are already very effective for such problems [Merz, 2004]. While computing $|Pop|^2$ distances per generation is a limitation in niching [Miller and Shaw, 1996; Smith *et al.*, 1993], this overhead is almost negligible in memetic algorithms where the local search procedure is much more time consuming. In multi-modal algorithms, the niching techniques also address intensification (by allowing a stable sub-populations on niches), but, in memetic algorithms, intensification can be ensured through local search. The diversity policy should only address diversification by permanently guiding the search to new regions. To sum up, our approach combines ideas from other evolutionary computation fields with new ideas specific for memetic algorithms in combinatorial optimization.

5.5 Experiments and results

5.5.1 Experimental conditions

The algorithm is tested on all the hard DIMACS benchmark instances presented in Section 1.2.3. Concerning the upper bound of the "easy graphs" from Section 1.2.3.1, they can be reached by Evo–Div in a short time.

5.5.1.1 Parameters

Parameter setting is not a particularly difficult task for Evo–Div—each parameter can be assigned a suitable value only by following explicit theoretical guidelines. By searching a perfect optimal value for each parameter, one could skew the results slightly more in Evo–Div's favor, but not enough to upset our main conclusions. We explain below how to reach a suitable (but not perfect) setting with limited tunning effort:

– Major genetic parameters: (i) population size is $|POP|$=20, (ii) the number of parents n is between 2 and 4, and (iii) target minimum spacing is $R = 10\%|V|$. We argued that the spacing strategy is intended for small population, and so, any population size between 10 and 30 could be used for reaching similar gloabal results. Previous work always used $n = 2$ parents, but we showed how to automatically set n according to instance specific information (i.e. the relation between k and $|V|$, see Section 5.3.1). Recall that the minimum spacing R was set to $10\%|V|$ using search space motivations (Section 5.4.2.1).

– The local search number of iterations is set to $maxIter = 100.000$ and the internal parameters are set to: $\alpha = 0.6$, $A = 10$ and $M_{\max} = 1000$ following previous work on Tabu Search (see also Section 5.2.2.1).

– Special case parameters : (i) *maxRejections*—the maximum number of rejected offspring before resorting to mutations, and (ii) mutation strength. By using *maxRejections* = 50, we are sure to respect the principle of preserving diversity without sacrificing quality because quality-deteriorating mutations are performed very rarely (i.e. in less than 0.1% of generations for any instance). Recall that the average number of rejected offspring per generation is normally between 0 and 3, as also indicated by $\frac{\text{crossovers–generations}}{\text{generations}}$ in Table 5.1. Only during the dispersion phase, we divide *maxRejections* by 10 in order to perform mutations more quickly to speed up leaving the current region. The mutation strength is set so as to perturb R vertices, enough to produce a mutated coloring outside the R-sphere of the initial coloring. Another guideline is to use a gradually-increasing strength—i.e. if the first offspring produced via mutation is rejected, the next one perturbs $2R$, and then $3R$, $4R$, etc (see also Section 5.4.2.2).

5.5.2 Standard general results with a time limit of 300 minutes

Graph (k^*)	k	Successes/Tries	Generations	Crossovers	Time[min]
$dsjc$500.1 (12)	12	10/10	301	428	1
$dsjc$500.5 (48)	48	10/10	370	373	7
$dsjc$500.9 (126)	126	8/10	1987	2157	63
$dsjc$1000.1 (20)	20	10/10	1658	2454	29
$dsjc$1000.5 (83)	83	9/10	2148	2439	136
$dsjc$1000.9 (223)	223	2/10	2872	3296	245
$dsjr$500.1c (85)	85	9/10	562	4156	93
$dsjr$500.5 (122)	122	8/10	1028	2230	36
r250.5 (65)	65	9/10	3175	6423	48
r1000.1c (98)	98	10/10	593	2240	98
r1000.5 (234)	238	9/10	953	1785	99
le450.25c (25)	25	10/10	10029	14648	90
le450.25d (25)	25	10/10	5316	7115	45
$flat$300.28.0 (28)	31	10/10	46	50	0
$flat$1000.76.0 (82)	82	10/10	1646	1884	110
$latin\_square$ (98)	100	1/10	585	973	42
C2000.5 (150[a])	148	4/10	5051	8953	2148
C4000.5 (280[b])	271	1/10	5960	29709	32142

Table 5.1: Detailed results of Evo–Div with a CPU time limit of 300 minutes on all hard instances. The algorithm reaches most of the best known results with a very high success rate (see Column 3)—the minimal value of k for which a solution was ever reported in the literature (i.e. k^*) is given in Column 1, in parenthesis.

[a]For this large graph, we used a time limit of 3 days; however, even 24 hours were enough for Evo-Div to find a first solution.

[b]For this exceptional large graph and for this very low k, we used a time limit of 30 days; however, legal 272-colorings can be reached in less than 10 days.

Table 5.1 presents the standard results of Evo–Div on the difficult instances with a time limit of 300 minutes. Columns 1 and 2 denotes the instance, i.e. the graph and the number of colors k. For each instance, this table reports the success rate over 10 independent runs (Column 3), the average number of generations required to find a solution (Column 4), the average number of crossovers (Column 5) and the average CPU time in minutes (last column). The total number of local search iterations is in close relation with the number of crossovers because the local search procedure (with $maxIter = 100000$) is applied once after each crossover.

Although a time limit of 5 hours is not very high for graph coloring, Evo–Div finds most of the best-known solutions from the literature—see also other algorithms in Table 5.3. If we consider the complete DIMACS benchmark from Section 1.2.3 (i.e. including the easy instances), Evo–Div matches the previously best-known results for 43 graphs out of 47 and it reaches a new coloring for ($dsjc1000.9$,$k = 223$). Only for 3 out of the 47 DIMACS instances, Evocol is below the best-known level—and one of these three upper bound, ($latin\_square,\ k = 98$), can actually be reached with a different Evo–Div variants, see Section 5.5.4 below.

We chose a time limit as the stopping condition because, in our case, machine-independent indicators are less meaningful and they can be easily misinterpreted. For example, a fixed limit on the number of generations would not take into account the amount of computational overhead introduced by a varying number of offspring rejections and distance calculations. Furthermore, the theoretical complexity of a generation, crossover, or iteration can be different from algorithm to algorithm; the comparison of such indicators could also be biased. Most recent algorithms [Malaguti *et al.*, 2008; Blöchliger and Zufferey, 2008; Hertz *et al.*, 2008; Galinier *et al.*, 2008][1] also use a time stopping condition and the best results are typically reached withing 10 hours. Our reported CPU times are obtained on a 2.8GHz Xeon processor using the C++ programming language compiled with the -O2 optimization option (gcc version 4.1.2 under Linux).

5.5.2.1 Evidence of Evo–Div Speed and Search Space Extensive Coverage

While we consider the 300 minutes time limit as a standard, we also show results with other time limits so as to better evaluate the algorithm. Table 5.2 presents the results using a time limit of 30 minutes (Columns 2-6) and 12 hours, respectively (Columns 7-11). The columns from this table have the same meaning as in the previous Table: the interpretation of Columns 2-6 (and Columns 7-11, respectively) is the same as for Columns 2-6 in Table 5.1.

Some interesting conclusions can be drawn from Table 5.2. Evo–Div can still find many best-known solutions within 30 minutes, which is very rapidly considering that the currently used execution times for graph coloring are usually hours or even days. To our knowledge, the smallest previous time limit (producing competitive results) is 1 hour for two articles published in 2008, see the results in Tables 1-4 in [Blöchliger and Zufferey, 2008] and Tables 1-3 in [Hertz *et al.*, 2008]. Evo–Div–30minutes finds many solutions not

[1]amacol check, mais plus important voir d'autres!!!

Graph (k^*)	Time limit: 30 minutes					Time limit: 12 hours				
	k	#hits	#gen	#cross	min	k	#hits	#gen	#cross	min
$DSJC1000.1$ (20)	20	7/10	1466	1667	23	20	10/10	1658	2454	29
$DSJC1000.5$ (83)	85	7/10	368	368	25	83	10/10	2943	3577	178
$DSJC1000.9$ (224)	225	1/10	298	304	28	223	3/10	4559	5252	400
$DSJC500.1$ (12)	12	10/10	301	428	1	12	10/10	301	428	1
$DSJC500.5$ (48)	48	10/10	370	373	7	48	10/10	370	373	7
$DSJC500.9$ (126)	126	2/10	453	514	15	126	10/10	3741	4319	125
$DSJR500.1c$ (85)	85	1/10	107	739	16	85	10/10	792	5936	136
$DSJR500.5$ (122)	122	5/10	422	593	10	122	9/10[a]	1659	4087	68
$flat1000.76.0$ (82)	83	1/10	401	402	29	82	10/10	1646	1884	110
$flat300.28.0$ (28)	31	10/10	46	50	0	31	10/10	46	50	0
$latin.square.10$ (98)	102	1/10	342	545	24	100	3/10	4189	6717	315
$le450.15c$ (15)	15	10/10	10	26	0	15	10/10	10	26	0
$le450.15d$ (15)	15	10/10	10	33	0	15	10/10	10	33	0
$le450.25c$ (25)	25	3/10	1660	1991	13	25	10/10	10029	14648	90
$le450.25d$ (25)	25	4/10	1593	1926	13	25	10/10	5316	7115	45
$r1000.1c$ (98)	98	4/10	149	311	13	98	10/10	593	2240	97
$r1000.5$ (234)	239	5/10	326	376	24	238	10/10	1661	2639	146
$r250.5$ (65)	65	6/10	650	1411	10	65	10/10	3961	10124	78

Table 5.2: Results of Evo–Div with two different time limits. In 30 minutes, Evo–Div still finds solutions not reached by other algorithms in hours. On the long run (12 hours), the diversity strategy assures a 100% success rates for many hard instances.

[a]For this case, 15 hours were required to attain 10/10 success rate

reached by the algorithms from any of these tables: ($DSJC1000.5$, $k = 85$), ($flat1000.76$, $k = 83$), ($le450.25c$, $k = 25$) and ($le450.25d$, $k = 25$). This first proves the effectiveness of the crossover operator, as the diversity policy is less active in a short run—e.g. the percentage of rejected offspring solutions (i.e. $\frac{\#\text{cross}-\#\text{gen}}{\#\text{gen}}$) is often very low.

The results on the long run (12 hours in Table 5.2) offer evidence that Evo–Div is capable of *search space extensive coverage*. Conventional algorithms may have difficulties in improving the performance by pushing the time limit beyond a certain threshold because they fail to exhaustively cover the search space—they can get blocked on plateaus, loop between certain areas, etc. By successfully solving such issues, our spacing policy can make Evo–Div reach 100% success rates if enough time is allowed.

One should be aware that the following rule is generally satisfied: if Evo–Div is capable to solve an instance, it solves it *systematically*. There are only two exceptions: (i) $DSJC1000.9$, which has been rarely solved with 223 colors (and it poses rather intensification challenges, see Section 5.5.4 below), and (ii) $latin\_square$, one of the only *three* Evo–Div "weak points". However, we are convinced that these "weak points" are not due to evolutionary diversity failures, but certain graphs require insisting on other points: (i) solving $flat300.28$ with $k = 28$ would need an intensification-oriented approach [Porumbel *et al.*, 2010] or an encodings based on partial solutions [Blöchliger and Zufferey, 2008;

Hertz *et al.*, 2008], (ii) $R1000.5$ could be solved with $k = 234$ only with column generation [Malaguti *et al.*, 2008] or other exact techniques [Prestwich, 2002], (iii) *latin_square* could be solved with $k = 98$ using special distributed Metropolis algorithms [Morgenstern, 1996]—however, see below that this graph can also be solved by a different Evo–Div version with a more generous time limit.

5.5.3 Comparison with state-of-the-art algorithms

Graph	χ/k^*	Evo–Div	Local Search Algorithms[a]				Hybrid Algorithms[a]					
		k (#hits)	ILS	VNS	PCol	VSS	DCNS	HGA	HEA	AmaCol	MMT	MCOL
		2009	2002	2003	2008	2008	1996	1996	1999	2008	2008	
		5 hours	1.6h	3h	10h	10h	>24h	>24h	≈3h	≈3h	≈10h	
*dsjc*500.1	?/12	12 (10/10)	12	—	12	12	—	—	—	12	12	12
*dsjc*500.5	?/48	48 (10/10)	49	49	48	48	49	49	48	48	48	48
*dsjc*500.9	?/126	126 (8/10)	126	—	126	126	—	—	—	126	127	126
*dsjc*1000.1	?/20	20 (10/10)	—	—	20	20	—	—	20	20	20	20
*dsjc*1000.5	?/83	83 (9/10)	89	90	88	87	89	84	83	84	83	83
*dsjc*1000.9	?/224 [223]	223 (2/10)	—	—	225	224	226	—	224	224	225	223
*dsjr*500.1*c*	84/85	85 (9/10)	—	—	85	85	85	85	—	86	85	85
*dsjr*500.5	122/122	122 (8/10)	124	—	126	125	123	130	—	125	122	122
*r*250.5	65/65	65 (9/10)	—	—	66	—	65	69	—	—	65	65
*r*1000.1*c*	98/98	98 (10/10)	—	—	98	—	98	99	—	—	98	98
*r*1000.5	234/234	238/237[b] (9/10)	—	—	248	—	241	268	—	—	234	245
*le*450.25*c*	25/25	25 (10/10)	26	—	25	26	25	25	26	26	25	25
*le*450.25*d*	25/25	25 (10/10)	26	—	25	26	25	25	—	26	25	25
*flat*300.28	28/28	31/29[b] (10/10)	31	31	28	28	31	33	31	31	31	29
*flat*1000.76	76/82	82 (10/10)	—	89	88	86	89	84	83	84	82	82
latin_square	?/98	100/98[b] (1/10)	99	—	—	—	98	106	—	104	101	99
*C*2000.5	?/150[148[c]]	148 (4/10)	—	—	—	—	150	153	—	—	—	148
*C*4000.5	?/280[272[c]]	271 (1/10)	—	—	—	—	—	280	—	—	—	272

Table 5.3: Best colorings reached by Evo–Div within 5 hours and results of the best performing coloring algorithms from the literature. The colorings of Evo–Div are publicly available at `http://www.info.univ-angers.fr/pub/porumbel/graphs/evodiv/`

[a]The acronyms of the algorithms can be found in Section 1.2.4, p. 18.

[b]Notice that the number of colors can be reduced for these graphs by using different Evo–Div versions, i.e. ($R1000.5$,k=237) was solved with a crossover that works only with independent sets; (flat300.28, $k = 29$) and (latin_square, $k = 98$) were solved using a longer TS chain, see Section 5.5.4.

[c]For the large graphs C2000.5 and C4000.5, we used a time limit of 3 and 30 days, respectively. Notice that the upper bounds in brackets are reported independently very recently (when this document was more than half written) in [Lü and Hao, 2010].

Table 5.3 compares Evo–Div (Column 3, reproducing Columns 2–3 from Table 5.1) with the best ten algorithms from the literature (Columns 4–13). For most instances, Evo–Div reaches very high success rates within a quite short time limit—the fifth line of Table 5.3 also shows the maximum time needed by other algorithms to reach certain solutions. For each algorithm, we also provide the publication year in Row 4 so as to have a rough idea of the computer generation that was used—very recent in half of the cases.

In principle, we use a unique Evo–Div version with a fixed setting of the parameters and with a rather short time limit. Only the very large graphs $C2000.5$ et $C4000.5$ require much more than 300 minutes but they are an exception and all known algorithms use time limits of days or weeks. The three upper bounds reached with different Evo-Div versions are clearly indicated. One should be aware that most individual columns from Table 5.3 actually *summarize* the best results of *several* algorithms or, at least, of several versions of the same algorithm.

5.5.4 Influence of diversity, crossover, and evaluation function

The goal of this section is to show the practical relevance of the most important ideas exploited by Evo–Div. The algorithm could still obtain good performance if one excludes certain components. For example, if one uses a less diversifying crossover, this weakness can be partially covered by the spacing strategy. For this reason, not all test instances are very sensitive to changes of theoretical design. However, we chose ten representative instances, showing the most important performance variation, and we test several Evo–Div versions on them.

Graph (k^*)	k	Evo–Div		*No-Div*		*Basic-Cross*		*R-5%*		*R-20%*		No-$\widetilde{f_{\text{eval}}}$	
		#hits	Min	#hits	Min	#hits	Min	#hits	Min	#hits	Min	#hits	Min
*DSJC*1000.1 (20)	20	10/10	29	0/10	–	4/10	211	9/10	37	10/10	31	10/10	26
*DSJC*1000.5 (83)	83	9/10	136	0/10	–	6/10	246	8/10	80	10/10	132	9/10	99
*DSJC*1000.9 (224)	223	2/10	245	1/10	110	2/10	220	2/10	183	0/10	–	0/10	–
*DSJR*500.1*c* (85)	85	9/10	93	0/10	–	10/10	55	0/10	–	10/10	21	3/10	76
*DSJR*500.5 (122)	122	8/10	36	1/10	4	10/10	25	1/10	4	7/10	65	2/10	108
*flat*1000.76.0 (82)	82	10/10	110	2/10	90	7/10	236	8/10	99	7/10	159	7/10	84
latinsquare (98)	100	1/10	42	0/10	–	0/10	–	0/10	–	0/10	–	1/10	211
*le*450.25*c* (25)	25	10/10	90	0/10	–	0/10	–	2/10	107	10/10	62	9/10	89
*le*450.25*d* (25)	25	10/10	45	1/10	42	0/10	–	1/10	16	10/10	86	10/10	87
*r*1000.5 (234)	238	9/10	99	1/10	19	1/10	250	2/10	27	6/10	76	0/10	–

Table 5.4: Comparison of the standard Evo–Div with five different versions obtained by excluding certain components. For each of these versions, we provide both the success rate (columns "#hits") and the time in minutes (columns "Min"). This experiment confirms many theoretical considerations presented throughout the paper.

Table 5.4 shows the success rate and the solving time obtained using Evo–Div and five other algorithm versions obtained by disabling certain components, as described below. The time limit is always 300 minutes, the processor and the machine are identical and the code source is also the same—except the lines that enable or disable a component. These results confirm our theoretical considerations on the components corresponding to the following Evo–Div versions:

1. *No-Div* (Columns 5 and 6): Evo–Div with no diversity strategy—i.e. all offspring is accepted and the worst individual is eliminated at the replacement stage. Comparing with Evo–Div (Columns 3 and 4), *No-Div* performs significantly worse. Even if it is still able to find some solutions, the stable success rates of Evo–Div are lost because diversity is no longer present;

2. *Basic-Cross* (Columns 7 and 8): Evo–Div with a crossover similar with the Greedy Partition Crossover [Galinier and Hao, 1999] (see also Section 5.3.2). Although *Basic-Cross* is able to eventually find 70% of the solutions reached with Evo-Div, importantly, *Basic-Cross* is much slower. Even by ignoring the failed 30% instances, *Basic-Cross* requires ten times more computing time (see $DSJC1000.1$) and this is the reason why it obtains low success rates. On the other hand, this crossover is still efficient for a difficult instance like ($DSJC1000.9$,$k = 223$);

3. *R-5%* (Columns 9 and 10): Evo–Div with a fixed target minimum spacing $R' = 5\%|V|$. The standard Evo–Div obtains systematically better results than this version. This confirms Section 5.4.2.1 where it is recommended to keep a distance of $R = 10\%|V|$ between any two individuals;

4. *R-20%* (Columns 11 and 12): Evo–Div with a fixed taget minimum spacing $R'' = 20\%|V|$. On half of the instances, this version fails or obtains low success rates. The results are quite good because $R'' > R$, and so, there is no lack of diversity in *R-20%*. On the other hand, there is too much diversification and not enough intensification. For example, it fails on ($DSJC1000.9$,$k = 223$), an instance that needs more intensification as it was solved even by *No-Div*. This confirms once again our recommendations of an optimum target minimum spacing of $R = 10\%|V|$;

5. *No-*$\widetilde{f_{\text{eval}}}$ (Columns 13 and 14): Evo–Div without the degree-based evaluation function $\widetilde{f_{\text{eval}}}$ in local search (see Section 5.2.2.1). The positive effect of the new function $\widetilde{f_{\text{eval}}}$ is more visible on graphs with high degree variation—especially the geometrical graphs (rX.Y and $DSJR$X.Y) in which the maximum degree can be with an order of magnitude higher than the minimum degree. It is normal that the degree-based differentiation of $\widetilde{f_{\text{eval}}}$ is less visible for graphs in which the degrees are more homogeneous (Leighton graphs, most of the random graphs). Note that $\widetilde{f_{\text{eval}}}$ has a positive influence on ($DSJC1000.9$,$k = 223$) as this graph has very high degrees. However, except this instance and the geometrical graphs, Evo–Div is still able to reach similar results with the classical evaluation function .

Finally, it is worth mentioning another two simple variants of Evo–Div that are able to improve the results on certain graphs. We have also tested Evo–Div with a longer TS chain ($maxIter$ = 10.000.000). By only changing this parameter, Evo–Div solved ($flat300.28$, $k = 30$) with 5/10 success rate within 5 hours, and it even reached one solution for the same graph using $k = 29$ colors; it also solves ($latin\_square$, $k = 98$) within 7.5 hours (by allowing a larger time limit of 24 hours, the success rate is more than 1/10). Furthermore, a version of our crossover working exclusively with independent sets leads to improved behaviour on some particular geometrical graphs. Using this crossover, Evo–Div reduced the solving time to seconds for the ($dsjr500.5$,$k = 85$) instance, and it also managed to color the very difficult $r1000.5$ graph with $k = 237$ colors.

5.6 Conclusions of the chapter

We described a new evolutionary hybrid algorithm (Evo–Div) that distinguishes itself by using a population spacing strategy that can both preserve and create diversity. We employ a Well-Informed Partition Crossover (WIPX) that exploits a large amount of information to choose the color classes used to construct offspring. We showed evidence that WIPX is able to help the population to quickly converge to solutions. On the other hand, the spacing strategy enables the population to successfully avoid premature convergence; by using many mechanisms to *create* diversity, it helps the search process to cover a very large area of the search space. Experiments show that the spacing strategy makes the memetic algorithm reach 100% success rate by using enough time.

The global results on all 47 DIMACS graphs (see also Section 5.5.3) are encouraging. Evo–Div finds almost all known best colorings and it reaches two upper bounds not known before this thesis: ($dsjc1000.9,\ k = 223$) and ($C4000.4,\ k = 271$). It might be interesting to notice that the legal coloring for ($latin\_square,\ k = 98$) was re-discovered for the first time in almost 15 years [Morgenstern, 1996]. In fact, Evo-Div reaches all best-known results *except for two graphs* ($r1000.5$ and $flat300.28$).

This shows the effectiveness of the memetic approach introduced in previous work (e.g. [Galinier and Hao, 1999; Malaguti *et al.*, 2008; Lü and Hao, 2010]), but also the contributions of Evo—Div in terms of diversity and crossover design. Although a detailed comparison goes beyond the purpose of this chapter, notice that ($dsjc1000.9,\ k = 223$) is only solved with TS-Div/Int (see Chapter 4), a method using in fact two algorithms with a total time limit of up to 100 hours [Porumbel *et al.*, 2010]. We also mention that this solution is also reported by a memetic algorithm (MemCol) that was independently developed by [Lü and Hao, 2010] and accepted for publication very recently. Even these new excellent results do not actually find any lower numbers of k compared to Evo–Div. While MemCol uses a similar time limit, it seems capable of visiting nearly 3 times more configurations for the same time period (the speed indicator "iterations per minute" of MemCol is 2-3 higher for certain large graphs like $dsjc1000.9$).

Finally, the spacing strategy is general enough to be used for other combinatorial optimization problems. The rejection mechanism is based only on calculating the minimum distance from the offspring to the population; the replacement operator does not need problem-specific operations (see Algorithm 5.3). To implement our memetic algorithm with diversity guarantee (see Algorithm 5.1), one only needs: a search space, a fitness function and a distance measure between individuals.

Chapter 6

Partition Distance: Graph Coloring Interpretation and Fast Calculation

In this chapter, we present in greater detail the partition distance and a fast computation method. A coloring is a particular case of a partition, and so, the partition distance has often been used for graph coloring. However, this distance has a far more general applicability than comparing colorings—partition distance algorithms are often employed in very diverse areas related to partitioning problems, e.g. clustering or image segmentation. We present a new *exact* algorithm for computing this distance and we discuss its applicability in different situations. If any of several conditions is met, the new algorithm is with an order of magnitude faster than the currently used method—i.e. straightforward reduction to the assignment problem solved via the Hungarian algorithm. The chapter is based on an article submitted in August, 2008 [Porumbel *et al.*, 2008].

Contents

6.1 Introduction

Let us denote by S a set and let k be a non-negative integer.[1] A k-partition of S (corresponding to a k-coloring of vertex set V) is a splitting of S into k non-overlapping classes (clusters, parts, blocks or cells) that cover all elements of S. Numerous practical applications dealing with data partitioning or clustering require computing the distance between two partitions. Previous articles proved that one can compute it in polynomial time—minimum $O(|S| + k^2)$ and maximum $O(|S| + k^3)$—using a reduction to the linear assignment problem.

We propose several conditions for which the partition distance can be computed in $O(|S|)$ time. In practical terms, this computation can be done in $O(|S|)$ time for any two relatively resembling partitions—i.e. any distance of less than $\frac{|S|}{5}$ can be computed in $O(|S|)$, except certain specially constructed cases. In addition, the algorithm can also be used to identify the maximal set of classes such that the similarity restricted to these classes is higher than a specific threshold.

6.1.1 Context and related studies

Given two $k-$partitions P_1 and P_2, the partition distance between P_1 and P_2 is defined as the minimum number of elements that need to be moved between classes of P_1 so that the resulting partition equals P_2. The similarity is defined as the maximum number of elements that do not require to be moved in order to obtain two equal partitions. We assume no restriction on the cardinality of a class: it can be empty and it can also be equal to S.

This definition of partition distance was first stated in 1965 by Régnier [Régnier, 1983] and the currently used computation methodology was presented by Day in 1981 [Day, 1981]. The method transforms the distance problem into the linear assignment problem on a $k \times k$ matrix. Solving the linear assignment problem by classical algorithms is known to take $O(k^2)$ time in the best case and up to $O(k^3)$ time in the worst case (i.e. with the Hungarian algorithm [Kühn, 1955]). A complete description of this methodology is available [Gusfield, 2002] and it is used by all recent studies ([Charon *et al.*, 2006; da Costa and Rao, 2004; Cardoso and Corte-Real, 2005; Konovalov *et al.*, 2005; Denœud and Guenoche, 2006; Talbi and Weinberg, 2007]) dealing with partition distances.

The Hungarian algorithm was a landmark result as it solved the assignment problem in polynomial time. Since the input data of this problem contains a $k \times k$ matrix, no algorithm can solve in less than $O(k^2)$ time. As such, the Hungarian method performs many other $O(k^2)$ operations without worsening its general complexity. In this thesis, the input data has only $O(|S|)$ elements (i.e. the two partitions) and one works with a similar

[1]For consistency with the set theory terminology, we denote the set by S, but it corresponds to V in the other chapters of the thesis.

matrix in which only $|S|$ elements (at maximum) are non zero. We believe that the best approach to compute the partition distance is not to straightforwardly apply the general Hungarian algorithm, mainly because one works with a sparse matrix. Algorithms for solving the assignment problem for sparse matrices are available [Carpaneto and Toth, 1983], but the possibility of reducing the complexity bellow $O(k^2)$ was not considered.

From a theoretical viewpoint, the partition distance satisfies several interesting properties. For instance, it is known that the partition distance constitutes a metric in the space of partitions [Cardoso and Corte-Real, 2005]. More in-depth studies [Charon *et al.*, 2006] show that, although the distance ranges from 0 to $|S|$, it can never reach $|S| - 1$ and more precise upper bounds are provided (e.g. $|S| - \left\lceil \frac{|S|}{k} \right\rceil$). A comparison between the distance function and other similar measures for partitions (e.g. the Rand index commonly used for comparing two data clusterings) is available [Denœud and Guenoche, 2006], showing the distribution of several indexes between close partitions. The distribution of the distances between random partitions is also studied, showing how one can interpret the value of a distance [Denœud, 2008]. Generalizations of this distance measure are also available in the literature [Berman *et al.*, 2007; Berger-Wolf *et al.*, 2007].

This last study [Denœud, 2008] could be particularly useful for graph coloring because it empirically observes that the average distance between two random partitions is in the region $65\%|S|$–$75\%|S|$ for $|S| = 100$ [Denœud, 2008, Table 1]. This gives important information on how to interpret distance values between colorings, i.e. a direct consequence is that two colorings distanced by $50\%|V|$ can not be considered "very far". Furthermore, the radius $R = 10\%|V|$ used to define the spheres in several of the developed heuristics represent indeed a "very small" value.

6.1.2 Why a new algorithm?

The algorithm introduced in this chapter computes the distance in $O(|S|)$ steps if any of several conditions is satisfied, e.g. if each class in P_1 shares with a class of P_2 at least half of the elements in each class. In addition, we show that the algorithm can be very useful in practice to compute *almost any* small distance. For illustration purposes, we also discuss experiments on a graph coloring algorithm that computes billions of small distances (less than $\frac{|S|}{5}$). The algorithm encounters very few distances (several hundreds per billion) that, although very small (less than $\frac{|S|}{5}$), require more than $O(|S|)$ steps.

The proposed algorithm can be used in numerous other applications concerned with close partitions, as for example those comparing a reference partition (a "gold standard") with a partition determined by an algorithm. A classical example is image segmentation, where a segmentation partition can be evaluated according to its distance from a correct/ideal segmentation [Cardoso and Corte-Real, 2005]. In biology, the distance is used to appreciate the difference (error) between a known partition of a population (a family structure) and a reconstruction based on genetic data [Konovalov *et al.*, 2005; Berger-Wolf *et al.*, 2007]. In clustering, one often obtains different partitions with different clustering algorithms and needs to find a consensus between them. To do this, one determines a central partition, i.e. a partition that minimizes the average distance to all other partitions [da Costa and Rao, 2004; Berman *et al.*, 2007]. If there is no much disagreement between the clustering algorithms, the partitions they produce will be sufficiently similar to be handled by the algorithm proposed in this chapter. Partition distances have been used in numerous other articles but we only touch on the most related publications in this chapter.

In the next section, we give a set of basic definitions. Section 6.3 discusses the conditions required for determining the distance in $O(|S|)$ steps. Section 6.4 extends the application of the proposed algorithm when some of these conditions are only partially satisfied. Section 6.5 discusses the application of the algorithm to graph colorings, which is followed by conclusions.

6.2 Distance definition

A $k-$partition P of a finite set $S = \{1, 2, \ldots, |S|\}$ is a function $P : S \to \{1, 2, \ldots, k\}$. It can also be defined as a set of classes $\{P^1, P^2, \ldots, P^k\}$ such that $\bigcup_{1\leq i\leq k} P^i = S$ and $P^i \cap P^j = \emptyset$, for all $i, j \in \{1, 2, \ldots, k\}$,$i \neq j$. The two definitions are equivalent since $P^i = \{x \in S|P(x) = i\}$; $P(x)$ identifies the number of the class of element x, and so, $x \in P^{P(x)}$ for all $x \in S$. If the P function is not surjective, some classes of the partition need to be empty.

Given two $k-$partitions P_1 and P_2 of S, we denote by $d(P_1, P_2)$ the distance between P_1 and P_2, i.e. the minimum number of elements that need to be moved between classes of P_1 so that the resulting partition becomes equal with P_2. The similarity $s(P_1, P_2)$ is a complementary measure of the distance denoting the maximum number of elements of P_1 that *do not need* to be moved in order to obtain equal partitions. The two measures satisfy the following equation:

$$s(P_1, P_2) + d(P_1, P_2) = |S|. \tag{6.1}$$

Alternatively, the distance can also be interpreted as the minimum number of elements one needs to erase from S such that the two partitions restricted to the set of remaining elements of S (denoted by S') are equal [Gusfield, 2002]. S' represents a set of elements that are shared by the two partitions and thus $|S'| = s(P_1, P_2)$.

To calculate the similarity $|S'|$, one needs to find the one-to-one correspondence σ : $\{1, 2, \ldots, k\} \to \{1, 2, \ldots, k\}$ (*assignment*) maximizing the sum:

$$s(P_1, P_2) = \max_{\sigma} \left(\sum_{1\leq i\leq k} T_{i,\sigma(i)} \right), \tag{6.2}$$

where T is the $k \times k$ similarity matrix $T(P_1, P_2)$ with elements:

$$T_{ij} = |P_1^i \cap P_2^j| \tag{6.3}$$

To determine the maximum of the sum in formula (6.2), one can solve a classical assignment problem. Indeed, most papers [Day, 1981; Gusfield, 2002; Charon *et al.*, 2006;

Konovalov *et al.*, 2005] suggest to apply on matrix T a classical Hungarian algorithm of time complexity between $O(k^2)$ and $O(k^3)$ [Kühn, 1955].

Normalized similarity and normalized distance Very often, it is useful to use the normalized values of the similarity and the distance: $s_{P_1,P_2} = \frac{s(P_1,P_2)}{|S|}$ and $d_{P_1,P_2} = \frac{d(P_1,P_2)}{|S|}$. These values represent a better indicator of the proportion of elements shared by two partitions, s_{P_1,P_2}, or that require to be moved, d_{P_1,P_2}. Clearly, the following formula holds for any S, P_1 and P_2: $d_{P_1,P_2} + s_{P_1,P_2} = 1$.

Note that since the distance is always strictly less than $|S|$, one can also normalize it with respect to other values, for example the maximum distance [Charon *et al.*, 2006]. However, the simple normalization above suffices for the purpose of this thesis.

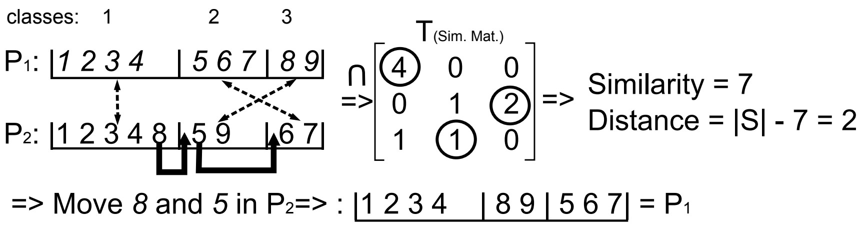
classes: 1 2 3
P1: 1 2 3 4 | 5 6 7 | 8 9
P2: 1 2 3 4 8 | 5 9 | 6 7
∩ =>
T(Sim. Mat.)
4 0 0
0 1 2
1 1 0
=> Similarity = 7
Distance = |S| - 7 = 2
=> Move 8 and 5 in P2=> : 1 2 3 4 | 8 9 | 5 6 7 = P1

Figure 6.1: An example of distance computation

Figure 6.1 above illustrates the process of distance calculation. There are two 3−partitions P_1 and P_2 of set $S = \{1, 2, \ldots, 9\}$ such that: $P_1^1 = \{1, 2, 3, 4\}$, $P_1^2 = \{5, 6, 7\}$, and $P_1^3 = \{8, 9\}$ denote the first partion, and $P_2^1 = \{1, 2, 3, 4, 8\}$, $P_2^2 = \{5, 9\}$ and $P_2^3 = \{6, 7\}$ denote the second. The similarity matrix T is computed with formula (6.3); the best assignment $\bar{\sigma}$, that maximizes the sum in formula (6.2), is defined by $\bar{\sigma}(1) = 1$, $\bar{\sigma}(2) = 3$ and $\bar{\sigma}(3) = 2$. We obtain $s(P_1, P_2) = T_{11} + T_{23} + T_{32} = 4 + 2 + 1 = 7$, and so, the distance is $d(P_1, P_2) = |S| - s(P_1, P_2) = 2$ (the normalized value is $d_{P_1,P_2} = \frac{2}{9} = 22\%|S|$). Indeed, if one changes the class of 2 elements (e.g. 8 and 5 in P_2), one partition is transformed into the other.

6.3 Distance computation

In this section, we describe the new $O(|S|)$ time algorithm and the necessary conditions for calculating the similarity—and implicitly the distance via (6.1)—of two given partitions P_1 and P_2 of a set S. The algorithm has two major steps: (i) construct the similarity matrix $T(P_1, P_2)$, and (ii) find the best $\bar{\sigma}$ in formula (6.2).

6.3.1 Similarity matrix T in $O(|S|)$ time

Our algorithm works on a $k \times k$ similarity matrix T, but only uses the following $|S|$ elements (at maximum): $T_{ij} = T_{P_1(x),P_2(x)}$, where $x \in S$. In fact, we know that all other elements of T are zero without calculating them—see formula 6.3—but such elements are not "touched". This construction step of T can be done in $O(|S|)$ time in three steps, as follows. First, one allocates memory for T (without any initialization); this requires $O(1)$

time because the allocation can be done at a block level, see below. In the second step, one goes through each $x \in S$ and initializes $T_{P_1(x),P_2(x)} = 0$—in $O(|S|)$ time. Finally, one goes through again each $x \in S$ by incrementing $T_{P_1(x),P_2(x)} := T_{P_1(x),P_2(x)} + 1$—$O(|S|)$ time again. In fact, the matrix structure is only used for indexing reasons, to quickly address the positions $T_{P_1(x),P_2(x)}$ with $x \in S$.

From now on, the values at positions $T_{P_1(x),P_2(x)}$ with $x \in S$ will be called *relevant.* The rest of the elements of T are considered irrelevant because the algorithm never needs them, neither for reading nor for writing.

The memory allocation of T should be done by finding a memory block of k^2 integers, but *without* initializing any value. Such allocations are usually done at a block level, and there is no theoretical requirement to perform a byte by byte checking on the k^2 block. We used the instruction `malloc(`$k^2\cdot$` sizeof(int))` that basically searches the list of free RAM blocks (chunks) to find one that is larger than the specified number of bytes—i.e. $k^2\cdot$ sizeof(int). In this manner, the time complexity of this operation depends on the fragmentation of the RAM memory, but not on the number of bytes to be allocated.[2]

6.3.2 Maximal assignment in $O(|S|)$ steps

This section discusses several conditions that enable our algorithm to compute the partition distance in $O(|S|)$ steps. We always consider that the input consists of an integer $|S|$ (such that, implicitly, S is $\{1, 2, \dots |S|\}$) and two vectors of integers denoting $P_1(x)$ and $P_2(x)$ for all $x \in S$. These two vectors can take values between 1 and k. We make use of the similarity matrix T from the previous section and the objective is to find a maximal assignment $\bar{\sigma}$, i.e. a bijective function $\bar{\sigma}$ maximizing the sum $\sum_{1\le i\le k} T_{i,\sigma(i)}$ in formula (6.2). For each of the presented conditions, the computation method is a Las Vegas $O(|S|)$ algorithm—it computes the distance or informs about the failure if the condition is not satisfied. In other words, the wasted time testing any condition is $O(|S|)$; it does not increase the total time complexity of a more general algorithm (i.e. Hungarian method) that could be needed afterward.

Theorem 6.1. *If for all $i \in \{1, 2, \dots, k\}$, there exists $j \in \{1, 2, \dots, k\}$ such that $T_{ij} > T_{ij_1}$ and $T_{ij} > T_{i_1j}$, for all $j_1 \neq j, i_1 \neq i$, then the partition distance can be determined in $O(|S|)$ time.*

Proof. In a first step, it is possible to compute in $O(|S|)$ the relevant elements of T (see Section 6.3.1) and also $T_{i\bar{\sigma}(i)}$, the unique maximum element of each row i. To determine these row-maximums, one goes only through the $O(|S|)$ relevant elements T and performs the following instruction: if T_{ij} is greater than the current maximum on row i (initially, this maximum is zero), the current maximum is updated to T_{ij}.

Since $T_{i\bar{\sigma}(i)}$ is a strict maximum on row i, any other mapping $\sigma : S \to S$ would lead to a lower sum $\sum_{1\le i\le k} T_{i,\sigma(i)}$. Checking that $\bar{\sigma}$ is bijective follows from the fact that if $\bar{\sigma}(i) = \bar{\sigma}(i') = j$, then both T_{ij} and $T_{i'j}$ represent the unique maximum of column j, and so, i and i' need to be the same. $\square$

[2]The interested reader can check a description of a malloc implementation—one of the most popular is Doug Lea's Malloc, see "A memory allocator" by Doug Lea.

The inequalities in the conditions from this theorem need to be strict, because otherwise one can not determine a bijective $\bar{\sigma}$, e.g. if $T = \begin{bmatrix} 2 & 1 \\ 2 & 1 \end{bmatrix}$. The case solved by this theorem can be seen as a dual of a specific Hungarian algorithm case in which the first step uncovers k mutually independent zeros (i.e. not lying in the same row or column). However, we compute the partition distance without converting the problem to a minimization problem (the Hungarian algorithm solves minimization problems) and without performing the $O(k \times k)$ row/column reductions needed by the Hungarian algorithm.

A similar condition could be expressed without mentioning the matrix T.

Corollary 6.2. *If for all $i \in \{1, 2, \ldots, k\}$, there exists $j \in \{1, 2, \ldots, k\}$ such that $|P_1^i \cap P_2^j| > \frac{|P_1^i|}{2}$ and $|P_1^i \cap P_2^j| > \frac{|P_2^j|}{2}$, then the partition distance can be determined in $O(|S|)$ time.*

Proof. The given hypothesis conditions represent a particular case of the conditions in Theorem 6.1. From (6.3), we have $\sum_{1\le\ell\le k} T_{i\ell} = \sum_{1\le\ell\le k} |P_1^i \cap P_2^\ell|$. Since all P_2^ℓ are disjoint and their union is S, $\sum_{1\le\ell\le k} |P_1^i \cap P_2^\ell| = |P_1^i \cap S| = |P_1^i|$. Therefore, for all $i \in \{1, 2, \ldots k\}$, we have

$$\sum_{1\le\ell\le k} T_{i\ell} = |P_1^i|. \tag{6.4}$$

By similar reasoning, we can conclude that:

$$\sum_{1\le\ell\le k} T_{\ell j} = |P_2^j|. \tag{6.5}$$

Using the hypothesis conditions, it follows that $T_{ij} > T_{ij_1}$ and $T_{ij} > T_{i_1 j}$, for all $j_1 \neq j, i_1 \neq i$ and the proof can be finished by using Theorem 6.1.

□

The main practical drawback of the conditions of this Corollary and of Theorem 6.1 is that, if there is a single row i on which they are *not* satisfied, the rest of the construction can not be used for determining the best assignment. The next theorem overcomes this issue and moreover, it can not be related to a dual of a step of the Hungarian algorithm. We show how one can determine the best assignment $i \xrightarrow{\bar{\sigma}} j$ on a row i by looking only at the elements on row i and column j—recall that the Hungarian algorithm returns only complete solutions and it takes no such intermediate (early) decisions on particular rows or columns.

Theorem 6.3. *If for row $i \in \{1, 2, \ldots, k\}$ there exists column $j \in \{1, 2, \ldots, k\}$ such that $T_{ij} \ge T_{ij_1} + T_{i_1 j}$ for all $j_1 \neq j, i_1 \neq i$, there exists a maximal assignment $\bar{\sigma}$ such that $\bar{\sigma}(i) = j$. If the number of rows i* not *satisfying this condition is bounded (i.e. less than $\sqrt[3]{|S|}$), the partition distance can be determined in $O(|S|)$ time.*

Proof. Following a very similar algorithm to the one in Theorem 6.1, one can determine matrix T and also the maximum value on each row and on each column. By going through the $O(|S|)$ relevant elements once again, one marks all maximum elements that are discovered on each row. Since only a marked row-maximum T_{ij} can satisfy $T_{ij} \geq T_{ij_1} + T_{i_1j}$ for all $j_1 \neq j, i_1 \neq i$, the algorithm just needs to check for each row-maximum T_{ij} that it is greater than $T_{i,-} + T_{-,j}$, where $T_{i,-}$ and $T_{-,j}$ are the second maximum values in row i and column j, respectively. We consider that $T_{i,-} = T_{ij}$ if and only if row i has at least two maximum value elements. Furthermore, determining the second maximum value of a row (or column, respectively) is very similar to determining the first maximum. Thus, the hypothesis condition can be checked for all rows in $O(|S|)$ time.

Let T_{ij} be an element marked by the above $O(|S|)$ procedure, such that $T_{ij} \geq T_{ij_1} + T_{i_1j}$ for all $j_1 \neq j, i_1 \neq i$. We need to show one can construct an maximum assignment by mapping i to j. Let σ be a maximal assignment. If $\sigma(i) = j$, then σ constitutes the searched assignment. Otherwise, let $j_1 = \sigma(i)$ and $i_1 = \sigma^{-1}(j)$. Using the hypothesis condition, one obtains:

$$T_{ij} + T_{i_1j_1} \geq T_{i_1j} + T_{ij_1} = T_{i_1\sigma(i_1)} + T_{i\sigma(i)} \tag{6.6}$$

By composing the transposition permutation (i, i_1) with σ, one obtains a new bijective mapping $\bar{\sigma}$ that differs from σ only on positions i and i_1, such that the values on these positions are switched, i.e. $\bar{\sigma}(i) = j$ and $\bar{\sigma}(i_1) = j_1$. The difference of value between assignments $\bar{\sigma}$ and σ (see (6.2)) is $T_{ij} + T_{i_1j_1} - (T_{i_1j} + T_{ij_1}) \geq 0$. Using (6.6), $\bar{\sigma}$ also needs to be a maximal assignment.

To summarize, an algorithm could establish a partial best assignment on all rows i that satisfy the hypothesis condition, regardless of the rows that do not satisfy this condition. This assignment on these rows is determined by mapping row i to a column j satisfying $T_{ij} \geq T_{ij_1} + T_{i_1j}$ for all $j_1 \neq j, i_1 \neq i$. If there are two rows i_1 and i_2 pointing to the same j, one can map i_1 to j and i_2 needs to be mapped to any other value on row i_2 (because all these values need to be zero).

The rest of the assignment can be constructed by applying the Hungarian algorithm on the remaining rows and columns. Under the given hypothesis condition, the number of unassigned $\bar{\sigma}$ elements is $k' = \left\lfloor \sqrt[3]{|S|} \right\rfloor$ in the worst case. To complete the assignment, one first marks the k' unassigned rows and the k' unassigned columns. A new $k' \times k'$ matrix is also allocated and initialized to zero in less than $O(|S|)$. Then, one goes through the relevant elements of T and copies into a new $k' \times k'$ matrix all elements situated at the intersection of a marked row and column. Finally, the Hungarian algorithm determines the maximum assignment value on this restricted matrix using maximum $O(k'^3) \leq O(|S|)$ operations, resulting in a total time complexity of $O(|S|)$ for the whole algorithm.

$\square$

A similar condition could be expressed without mentioning the matrix T, in a simpler manner.

Corollary 6.4. *If for all $i \in \{1, 2, \ldots, k\}$, there exists $j \in \{1, 2, \ldots, k\}$ such that $|P_1^i \cap P_2^j| \geq \frac{|P_1^i \cup P_2^j|}{2}$, then the partition distance can be computed in $O(|S|)$ steps.*

Proof. This proposition follows from equations (6.4) and (6.5) as it becomes a particular case of Theorem 6.3. However, this corollary also has the advantage that it is very easy to implement because $|P_1^i \cup P_2^j| = |P_1^i| + |P_2^j| - |P_1^i \cap P_2^j|$ and $|P_1^i|$ and $|P_2^j|$ can be easily determined.

□

Regarding the most general proofs of this section, note that Theorem 6.1 and Theorem 6.3 do not result one from another. For example, if $T = \begin{bmatrix} 3 & 2 \\ 2 & 3 \end{bmatrix}$, only Theorem 6.1 can be used; if $T = \begin{bmatrix} 2 & 2 \\ 0 & 2 \end{bmatrix}$, one should use Theorem 6.3. In the next section, we show how Theorem 6.3 and Corollary 6.4 can also be used in practice to construct only a part of the solution.

6.4 Extensions

In case the hypothesis conditions of Theorem 6.3 or Corollary 6.4 only hold for a restricted set of rows $i \in \{1, 2, \ldots, k\}$, we can still perform important time complexity reductions. First, let us prove the following proposition:

$$|P_1^i \cap P_2^j| \leq \frac{|P_1^i \cup P_2^j|}{2} \text{ for all } i, j \in \{1, 2, \ldots, k\} \Longrightarrow s(P_1, P_2) \leq \frac{2}{3}|S|. \tag{6.7}$$

Proof. Let $\bar{\sigma}'$ be a maximal assignment and $j = \bar{\sigma}'(i)$, where i is any row. One can write $T_{ij} \leq \frac{|P_1^i \cup P_2^j|}{2}$ as $T_{ij} \leq \frac{|P_1^i| + |P_2^j| - |P_1^i \cap P_2^j|}{2}$, or $3T_{ij} \leq |P_1^i| + |P_2^j|$. Making the sum over all rows i, one obtains $3\sum_{1 \leq i \leq k} T_{i\bar{\sigma}'(i)} \leq \sum_{1 \leq i \leq k} |P_1^i| + \sum_{1 \leq j \leq k} |P_2^j| = 2|S|$, which proves (6.7). □

We use the conditions of Corollary 6.4 only for a greater readability, but the same result could be derived for the conditions of Theorem 6.3, i.e. if $T_{ij} < T_{ij_1} + T_{i_1 j}$ for all $j_1 \neq j, i_1 \neq i$, then $|P_1^i \cap P_2^j| \leq \frac{|P_1^i \cup P_2^j|}{2}$ is also satisfied—see (6.4) and (6.5).

Now, we present the actual reduction of the $s(P_1, P_2)$ computation into smaller pieces. We divide S into two subsets A and B such that only the computation on B requires an algorithm of higher complexity. Let us denote by I the set of elements i for which there is $j_i \in \{1, 2, \ldots, k\}$ such that $|P_1^i \cap P_2^{j_i}| > \frac{|P_1^i \cup P_2^{j_i}|}{2}$. We write $J = \{j \in S|$ there exists $i \in I$ s.t.$j = j_i\}$ and let

$$A = \bigcup_{i \in I} P_1^i \cup P_2^{j_i} \tag{6.8}$$

and $B = S - A$. Using A in the reasoning of Theorem 6.4 or 6.3, one finds there exists a maximal assignment $\bar{\sigma}$ satisfying $\bar{\sigma}(i) = j_i$, for all $i \in I$. Since J is the image of I through the bijective $\bar{\sigma}$, then $\{1, 2, \ldots, k\} - J$ is the image of $\{1, 2, \ldots, k\} - I$. The rest of $\bar{\sigma}$ can be constructed only using rows and columns from these two sets, which contain values generated only by classes of B (i.e. subsets $P_1^i, P_2^j \subseteq B$).

Writing $s(P_1, P_2)|_X$ the similarity between partitions P_1 and P_2 restricted to set $X \subset S$, we obtain $s(P_1, P_2) = s(P_1, P_2)|_A + s(P_1, P_2)|_B$. Since no confusion arises, we can simply write: $|S| \times s_{P_1,P_2} = |A| \times s_{P_1,P_2}|_A + |B| \times s_{P_1,P_2}|_B$ and we can even omit the index $_{P_1,P_2}$:

$$s_S = \frac{|A|}{|S|} s_A + \frac{|B|}{|S|} s_B,$$

where s_X is the normalized similarity between P_1 and P_2 restricted to set X.

The sets A and B can be directly determined from set I using (6.8) and I can be determined in $O(|S|)$ time, following the reasoning of Theorem 6.3. Furthermore, s_A can be determined in $O(|A|) < O(|S|)$ as explained in Section 6.3.2; s_B can be determined in maximum $O((k - |I|)^3)$ using the Hungarian algorithm. To summarize, the total time complexity of computing the similarity this way is $O(|S|) + O((k - |I|)^3)$ at maximum.

Using (6.7), we obtain $s_B \leq \frac{2}{3}$. This means that if the total similarity is high (i.e. for example $s_S > 0.9$), S can be split in two parts:

1. A, on which the normalized similarity s_A is very high (e.g. $s_A > s_S > 0.9$) and can be computed in $O(|S|)$.

2. B, on which the normalized similarity is *much* lower $s_B \leq \frac{2}{3}$ and can not be computed in $O(|S|)$.

In case the total similarity of P_1 and P_2 is high, even if we cannot always compute it in $O(|S|)$ time, we can always identify the part of S where the matching is stronger (i.e. A) in $O(|S|)$ time. This could be particularly useful for applications that only need to find the best class matches between two partitions.

6.5 A graph coloring case study

In this section we present numerical conclusions observed in the context of a graph coloring local search (TS-Div, see Section 4.2) that computes billions of distances to guide itself through the search space. This algorithm tries to stay at a certain distance from certain reference colorings and the time complexity of distance computing is crucial—e.g. for certain graphs, a $O(k^3)$ algorithm can make the process at least 100 times slower. The details of the heuristic are not essential here, but we only discuss a statistic of the computed distances.

We consider one billion *small* distances computed by this algorithm while solving two standard coloring instances (DIMACS instances ($dsjc1000.1$, $k = 20$) and ($dsjc1000.5$, $k = 86$) with 1000 vertices). In fact, many distances computed by TS-Div were *small* because they represent distances between close positions in a series of neighboring colorings; two neighboring coloring differ only by the color of a single vertex. However, even if there are also greater distances, we only count in this statistics the pairs (P_1, P_2) satisfying $d(P_1, P_2) < \frac{|S|}{5}$. For each distance calculation, TS-Div applies our partition distance algorithm following this methodology:

1. If the condition in Corollary 6.4 is satisfied, the algorithm simply computes the correct distance in $O(|S|)$ time.

2. Otherwise, the algorithm detects that the condition is not satisfied (it returns $IMPOSSIBLE$ in $O(|S|)$ time, see Algorithm 6.1, p. 104) and the $O(k^3)$ Hungarian algorithm is executed.

The first step was sufficient for more than 99.99% cases. More precisely, in more than 10^9 computed distances, we found less than 10^3 (i.e. 737 when $k = 20$ and 880 for the $k = 86$ case) pairs (P_1, P_2) for which $d(P_1, P_2) < \frac{|S|}{5}$ but the hypothesis condition in Corollary 6.4 is not satisfied. If we consider even smaller distances (more exactly, only pairs (P_1, P_2) such that $d(P_1, P_2) < \frac{|S|}{10}$) the $O(|S|)$ time algorithm is sufficient for all practical cases we encountered.

The explanation of this practical success lies in the fact that the similarity restricted to different subsets of S presents quite homogeneous values in practice. Thus, if the total similarity is high (i.e. $s_{P_1,P_2} > 0.9$ equivalent to $d(P_1, P_2) < \frac{|S|}{10}$, the s_X values are quite close to 0.9 for most classes X of the partitions. Thus, the cardinal of set B (on which $s_B < \frac{2}{3}$, and the conditions from section 6.3.2 are not met) is very limited, usually B is empty in practice.

However, theoretically one can still construct a counter-example to this by taking two partitions such that $P_1^1 = P_2^1$ and $|P_1^1| = 0.9|S|$. In this case $s_{P_1,P_2} \geq 0.9$, but the two partitions, which are very similar on $A = P_1^1$, can be totally different on $B = S - A$. The most difficult part is the computation of the similarity restricted to B; only this one may still require between $O(k^2)$ and $O(k^3)$ time.

6.6 Concluding remarks

This chapter introduced a very fast algorithm for computing the distance between two close partitions P_1 and P_2 of a set S (for graph coloring the set S is actually the vertex set V). If any of several proposed conditions is verified, then the distance value can be computed in $O(|S|)$ time—otherwise it is possible to inform about the failure in $O(|S|)$ time. In the coloring context, this algorithm makes and important difference, because the conventional method would require at least $O(|S| + k^2)$ time—for an instance like $(dsjc1000.9,\ k = 223)$, a number of operations of about $|S| + k^2 = 1000 + 223^2 \approx 50.000$ is much more costly than $|S| = 1000$.

If no proposed condition is satisfied, our algorithm returns “impossible” in $O(|S|)$ time. In this situation, the Hungarian algorithm is finally executed, but there is no complexity overhead introduced by our algorithm, i.e. $O(|S|)$ does not matter in a complexity sum of the form $O(|S| + k^2) + O(|S|) = O(|S| + k^2)$. Moreover, the proposed algorithm can also be useful even if the required conditions are not totally satisfied, as it is explained in Section 6.4. In such a situation, the algorithm can be used to identify the subset of S on which the matching is stronger, i.e. where the normalized similarity is at least $\frac{2}{3}$. Finally, certain ideas can be very useful for solving general assignment problems defined with sparse matrices.

6.6.1 A complete pseudocode example of the algorithm

The algorithm we present in this appendix (p. 104) corresponds to Corollary 6.4 and, as one can see, it is quite simple. By detailing it, one can also implement a similar version corresponding to any other theorem in this chapter. The number of classes does not need to be the same for P_1 and P_2 because k is not given in the input; it is computed as the maximum number of classes in any partition.

Algorithm 6.1: Algorithm For Computing The Partition Distance, corresponding to Corollary 6.4.

```
Inputs:
  • $|S|$, so that $S = \{1, 2, \dots, |S|\}$
  • $P_1$ and $P_2$ as $|S|$-vectors (i.e. position $x$ in $P_1$ represent $P_1(x)$, for all $x \in S$)
Return value:
  • the distance $\Sigma$, if the condition in Corollary 6.4 is satisfied
  • *IMPOSSIBLE*, otherwise.
Begin
  1. init $\Sigma = 0$
  2. init $k$ = maximum value in vectors $P_1$ and $P_2$
  3. allocate the $k \times k$ matrix $T$ (without filling any element)
  4. init $k-$vectors $M$ and $\bar{\sigma}$ to 0 (the maximum on each row and the maximal
     assignment that is constructed)
  5. init $k-$vectors $|P_1|$ and $|P_2|$ to 0 (denoting the cardinal of each class)
  6. FOR $x = 1$ TO $|S|$
       • set $i = P_1(x)$ and $j = P_2(x)$
       • set $T_{ij} = 0$
  7. FOR $x = 1$ TO $|S|$
       • set $i = P_1(x)$ and $j = P_2(x)$
       • increment $T_{ij}$, $|P_1^i|$, $|P_2^j|$
       • IF $T_{ij} > M_i$, THEN set $M_i = T_{ij}$ and $\bar{\sigma}(i) = j$.
  8. FOR $i = 1$ TO $k$
       • IF $M_i = 0$, THEN CONTINUE (with next $i$)
       • IF $3M_i \leq |P_1^i| + |P_2^{\bar{\sigma}(i)}|$, THEN RETURN *IMPOSSIBLE*
       • set $\Sigma = \Sigma + T_{i,\bar{\sigma}(i)}$
  9. RETURN $\Sigma$
End
```

General Conclusion

The idea of using a search space metric to obtain a *position-guided search* seems, to a certain extent, overlooked in combinatorial optimization. Related distance-based ideas *have* been applied only for more specific optimization purposes (e.g. distances-preserving crossovers, crowding in multi-modal continuous optimization, fitness distance correlation, etc.). However, we showed that distances can be used in a more systematic way, either for understanding the search space, or to induce a navigation/orientation "compass" in the optimization process. Such techniques can be seen as part of an emerging new area of "learning methods in optimization". Indeed, heuristic algorithms can be substantially improved by collecting information ("on-line" or "off-line") about the search space structure and about the patterns of visited configurations.

Along the thesis, we (try to) extract and exploit the most relevant and meaningful information in the design of all components, so as to make all algorithms as "well-informed" as possible. Four algorithms (RCTS, TS-Div, TS-Int, Evo-Div) have been tested on the challenging experimental framework of the graph coloring problem. Indeed, this is one of the most famous and most studied NP-complete problems—a dozen of new algorithms have been published only during this 3-years time thesis. Although the number of algorithms is continually growing, we have reached almost all best-known upper bounds and we have even found certain new legal colorings for the first time.[3]

The contributions of this thesis can be classified as follows. In the introduction, we reviewed the main issues related to learning in optimization, and we presented the best graph coloring algorithms. In Chapter 2, we started by presented a simple coloring Tabu Search algorithm that has been improved with new "well-informed" evaluation functions. By coupling a few additional algorithmic techniques—i.e. a reactive Tabu list—this "lightweight" Tabu Search can reach very competitive results compared to the best ten algorithms from the literature.

By investigating the trajectory of the basic Tabu Search, we presented in Chapter 3 the clustering hypothesis: the high-quality visited configurations are not uniformly scattered in the search space, but grouped in clusters that can be confined in spheres of radius $R = 10\%|V|$. In the rest of the thesis, we considered that two colorings distanced by less than $10\%|V|$ are "too close". We have defined the radius R based on the *partition distance* measure (a search space distance, equal to the minimum number of neighborhood transitions between colorings).

In Chapter 4 we presented two position-guided algorithms that are based on the search space distance and on the notion of sphere. The first one (TS-Div) is "guided" toward diversification. It performs a coarse-grained recording of its own trajectory by recording a limited number of visited spheres. In this manner, it is "aware" when the search process is entering an already-visited sphere. When this happens, it acts so as to induce more diversification in the following moves, in order to quickly leave the sphere. This guarantees

[3]For the 15 years old DIMACS graphs, the standard benchmark in graph coloring. All mentioned legal colorings are available on-line for further study: `info.univ-angers.fr/pub/porumbel/graphs/bestcol/`

that the local search does not perform redundant explorations in the long run. Since the basic Tabu list forbids redundant explorations in the short run, TS-Div is able to cover (more) properly the search space.

The TS-Int algorithm is guided for intensification, so as to thoroughly explore a given limited perimeter, a specific promising region. Starting from an input configuration, it launches several local search processes that can run *only inside the sphere* of the start configuration. Such a process is stopped as soon as it leaves the sphere, and the "sphere exit point" is inserted in a queue of "start configurations", i.e. configurations whose spheres will be explored later. When the processes launched from a start configuration do no longer find distanced high-quality sphere exit points (the sphere investigation is complete), the next start configuration is considered and TS-Int continues with another sphere investigation on it. The result is a "breath-first Tabu search" that seems to be able to find any solution within less than $30\%|V|$ distance from the starting point (with 100% empirical success rate).

An evolutionary approach is presented in Chapter 5. One contribution is the introduction of a spacing strategy that forbids the population from keeping at the same time two close individuals in the population. Using distance-based reactive dispersion mechanisms, the spacing among individuals is always kept at "healthy levels". This way, premature convergence can be avoided; the resulting algorithm (Evo–Div) is actually able to continually create useful diversity without quality sacrifices. A well-informed crossover is also presented; it uses several features of the parent color classes in order to decide the most appropriate information that is passed to offspring.

Finally, the distance function is studied in detail in the last chapter, providing more insight into how to interpret distance values. We also introduce a Las Vegas exact algorithm that, if one of several conditions is verified, reduces the distance calculation time from $O(|V| + k^3)$ to $O(|V| + k)$. Besides the speed-up brought to TS-Div, this algorithm can also be used in numerous other practical applications dealing with distances between close partitions.

Prospects

A distance measure can be very useful to capture several notions that can help understanding the search space, in particular: search space position, close and distant candidate solutions, or spatial distribution. We think that distance measures can be employed in a more systematic way in optimization, both for understanding the search space, and also to equip a search process with a "compass" of navigation/orientation through this space.

Although these ideas were validated only on the graph coloring problem, certain strategies are very general and could also be tested on other problems. For example, how can one know if a local search process covers its search space uniformly, or if it makes many redundant explorations? The coarse-grained trajectory recording of TS–Div offers a potential answer: by recording a restricted set of spheres, one can indeed inspect the trajectory of the search process and clarify this kind of questions. The only condition to carry out that is to be able to calculate a neighborhood distance, i.e. a distance which indicates the (minimum) number of neighborhood transitions between two configurations—see several examples in the Section 4.5.

Concerning the evolutionary approach, it would be very interesting to formalize a generic strategy of diversity: we think that the spacing control ideas can be directly used in other evolutionary hybrid algorithms. Obviously, to obtain at the best practical results, it is always essential to study the details and the specific characteristics of the problem, so as to adapt the approach. However, similar ideas have been already tested successfully in other evolutionary communities (e.g. multimodal continuous optimization), and thus, we think that more progress can be expected.

Furthermore, a more in-depth work could couple the ideas concerning the local search trajectory recording with the spacing control in the evolutionary algorithm. While the spacing control ensures that two very close individuals can not be simultaneously in the population, it does not ensure similar properties on individuals belonging to different generations—such problem could be avoided by recording the spheres of all individuals at all generations (their number should be limited, compared to the number of all individuals ever visited by the evolutionary search).

List of figures

List of tables

List of algorithms

References

[Avanthay *et al.*, 2003] cited page 14, 18, 36
C. Avanthay, A. Hertz, and N. Zufferey. A variable neighborhood search for graph coloring. *European Journal of Operational Research*, 151(2):379–388, 2003.

[Barnier and Brisset, 2004] cited page 13
N. Barnier and P. Brisset. Graph coloring for air traffic flow management. *Annals of Operations Research*, 130(1):163–178, 2004.

[Battiti and Tecchiolli, 1994] cited page 9
R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA journal on computing*, 6:126–126, 1994.

[Battiti *et al.*, 2008] cited page 6, 8, 8, 9, 30, 40
R. Battiti, R Brunato, and F. Mascia. *Reactive Search and Intelligent Optimization.* Springer, 2008.

[Beasley *et al.*, 1993] cited page 83, 83, 83
D. Beasley, D.R. Bull, and R.R. Martin. A sequential niche technique for multimodal function optimization. *Evolutionary Computation*, 1(2):101–125, 1993.

[Bellare *et al.*, 1998] cited page 12
M. Bellare, O. Goldreich, and M. Sudan. Free bits, PCPs, and nonapproximability-towards tight results. *SIAM Journal on Computing*, 27(3):804, 1998.

[Berger-Wolf *et al.*, 2007] cited page 94, 94
T.Y. Berger-Wolf, S.I. Sheikh, B. DasGupta, M.V. Ashley, I.C. Caballero, W. Chaovalitwongse, and S.L. Putrevu. Reconstructing sibling relationships in wild populations. *Bioinformatics*, 23(13):i49, 2007.

[Berman *et al.*, 2007] cited page 94, 94
P. Berman, B. DasGupta, M.Y. Kao, and J. Wang. On constructing an optimal consensus clustering from multiple clusterings. *Information Processing Letters*, 104(4):137–145, 2007.

[Blöchliger, 2005] cited page 12
I. Blöchliger. *Suboptimal Colorings and Solution of Large Chromatic Scheduling Problems.* PhD thesis, École Polytechnique Fédérale de Lausanne, 2005.

[Blöchliger and Zufferey, 2008] cited page 13, 13, 14, 14, 14, 18, 22, 23, 30, 30, 34, 36, 41, 65, 65, 65, 67, 86, 86, 87
I. Blöchliger and N. Zufferey. A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Computers and Operations Research*, 35(3):960–975, 2008.

[Boese *et al.*, 1994] cited page 9
KD Boese, AB Kahng, and S. Muddu. A new adaptive multi-start technique for combinatorial global optimizations. *Operations Research Letters*, 16(2):101–113, 1994.

[Bouziri *et al.*, 2008] cited page 19
H. Bouziri, E.G. Talbi, and K. Mellouli. A cooperative search method for the k-coloring problem. *Journal of Mathematical Modelling and Algorithms*, 7(2):125–142, 2008.

[Boyan *et al.*, 2000] cited page 8, 8
J. Boyan, W. Buntine, and A. Jagota. Statistical machine learning for large-scale optimization. *Neural Computing Surveys*, 3(1):1–58, 2000.

[Brelaz, 1979] cited page 13, 13
D. Brelaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.

[Brown, 1972] cited page 13
J.R. Brown. Chromatic scheduling and the chromatic number problem. *Management Science*, 19(4):456–463, 1972.

[Burke *et al.*, 1994] cited page 13
E. K. Burke, D. G. Elliman, and R. F. Weare. A university timetabling system based on graph colouring and constraint manipulation. *Journal of Research on Computing in Education*, 27(1):1–18, 1994.

[Cardoso and Corte-Real, 2005] cited page 93, 94, 94
J.S. Cardoso and L. Corte-Real. Toward a generic evaluation of image segmentation. *IEEE Transactions on Image Processing*, 14(11):1773–1782, 2005.

[Carpaneto and Toth, 1983] cited page 93
G. Carpaneto and P. Toth. Algorithm for the solution of the assignment problem for sparse matrices. *Computing*, 31(1):83–94, 1983.

[Cedeño and Vemuri, 1999] cited page 83, 83, 83
W. Cedeño and V.R. Vemuri. Analysis of speciation and niching in the multi-niche crowding ga. *Theoretical Computer Science*, 229(1):177, 1999.

[Chaitin, 1982] cited page 13
G. J. Chaitin. Register allocation & spilling via graph coloring. In *Proceedings of the 1982 SIGPLAN symposium on Compiler construction*, pages 98–105, New York, NY, USA, 1982. ACM.

[Chams *et al.*, 1987] cited page 14
M. Chams, A. Hertz, and D. de Werra. Some experiments with simulated annealing for coloring graphs. *European Journal of Operational Research*, 32(2):260–266, 1987.

[Charon *et al.*, 2006] cited page 93, 94, 95, 96
I. Charon, L. Denoeud, A. Guenoche, and O. Hudry. Maximum transfer distance between partitions. *Journal of Classification*, 23(1):103–121, 2006.

[Chiarandini and Stützle, 2002] cited page 14, 18, 36
M. Chiarandini and T Stützle. An application of iterated local search to graph coloring. In D. S. Johnson et al., editors, *Computational Symposium on Graph Coloring and its Generalizations*, pages 112–125, 2002.

[Chiarandini *et al.*, 2007] cited page 12, 23
M. Chiarandini, I. Dumitrescu, and T. Stützle. Stochastic local search algorithms for the graph colouring 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.

[Chiarandini, 2005] cited page 12
M. Chiarandini. *Stochastic Local Search Methods for Highly Constrained Combinatorial Optimisation Problems.* PhD thesis, Darmstadt University of Technology, Germany,, 2005.

[Chow and Hennessy, 1990] cited page 13
F.C. Chow and J.L. Hennessy. The priority-based register allocation. *ACM Transactions on Programming Languages and Systems*, 12(4):501–536, 1990.

[Christofides, 1971] cited page 13
N. Christofides. An algorithm for the chromatic number of a graph. *The Computer Journal*, 14(1):38–39, 1971.

[Coello *et al.*, 2004] cited page 82
C.A.C. Coello, G.T. Pulido, and M.S. Lechuga. Handling multiple objectives with particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3), 2004.

[Costa *et al.*, 1995] cited page 71
D. Costa, A. Hertz, and C. Dubuis. Embedding a sequential procedure within an evolutionary algorithm for coloring problems in graphs. *Journal of Heuristics*, 1(1):105–128, 1995.

[Culberson and Gent, 2001] cited page 39, 40
J.C. Culberson and I. Gent. Frozen development in graph coloring. *Theoretical Computer Science*, 265:227–264, 2001.

[Culberson and Luo, 1996] cited page 13, 16, 64
J.C. Culberson and F. Luo. Exploring the k-colorable landscape with iterated greedy. In *Cliques, Coloring, and Satisfiability Second DIMACS Implementation Challenge* [Johnson and Trick, 1996], pages 345–284.

[da Costa and Rao, 2004] cited page 93, 94
J.F.P. da Costa and P.R. Rao. Central partition for a partition-distance and strong pattern graphs. *REVSTAT–Statistical Journal*, 2(2):127–143, 2004.

[Day, 1981] cited page 93, 95
W.H.E. Day. The complexity of computing metric distances between partitions. *Mathematical Social Sciences*, 1:269–287, 1981.

[De Jong, 1975] cited page 83, 83
K.A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan Ann Arbor, MI, USA, 1975.

[De Werra and Gay, 1994] cited page 13
D. De Werra and Y. Gay. Chromatic scheduling and frequency assignment. *Discrete applied mathematics*, 49(1-3):165–174, 1994.

[de Werra, 1985] cited page 13
D. de Werra. An introduction to timetabling. *European Journal of Operational Research*, 19(2):151–162, 1985.

[Deb and Goldberg, 1989] cited page 83, 83
K. Deb and D. E. Goldberg. An investigation of niche and species formation in genetic function optimization. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 42–50, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[Deb *et al.*, 2002] cited page 82, 82
K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[Denœud and Guenoche, 2006] cited page 93, 94
L. Denœud and A. Guenoche. Comparison of distance indices between partitions. In V. Batagelj et al., editors, *Data Science and Classification*, pages 21–28. Springer, Berlin, Germany, 2006.

[Denœud, 2008] cited page 94, 94, 94
L. Denœud. Transfer distance between partitions. *Data Science and Classification*, 2(3):279–294, 2008.

[Devarenne *et al.*, 2006] cited page 14, 18, 23, 28, 30, 30, 36
I. Devarenne, Mabed H., and A. Caminada. Intelligent neighborhood exploration in

local search heuristics. In *Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence*, pages 144–150. IEEE Computer Society, 2006.

[Devarenne, 2007] cited page 12
I. Devarenne. *Études en recherche locale adaptative pour l'optimisation combinatoire.* PhD thesis, Université de Technologie de Belfort Montbéliard, France, 2007.

[Dorne and Hao, 1995] cited page 13
R. Dorne and J.K. Hao. An evolutionary approach for frequency assignment in cellular radio networks. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 539–544, 1995.

[Dorne and Hao, 1998a] cited page 14, 14, 14, 19, 22, 23, 34, 71, 76
R. Dorne and J.K. Hao. A new genetic local search algorithm for graph coloring. In *PPSN 98*, volume 1498 of *LNCS*, pages 745–754. Springer, 1998.

[Dorne and Hao, 1998b] cited page 13, 14, 23, 23, 30
R. Dorne and J.K. Hao. Tabu search for graph coloring, t-colorings and set t-colorings. In S. Voss et al., editors, *Meta-Heuristics Advances and Trends in Local Search Paradigms for Optimization*, pages 77–92. Kluwer, 1998.

[Dorne, 1998] cited page 12
R. Dorne. *Méthodes heuristiques pour la coloration, la T-coloration et l'affectation de fréquences.* PhD thesis, Université de de Montpellier, France, 1998.

[Du and Pardalos, 2007] cited page 39, 57
D. Du and P.M. Pardalos. *Handbook of Combinatorial Optimization.* Springer, 2007.

[Falkenauer, 1998] cited page 70, 74
E. Falkenauer. *Genetic algorithms and grouping problems.* John Wiley & Sons, Inc. New York, NY, USA, 1998.

[Fleurent and Ferland, 1996a] cited page 13, 14, 14, 22, 23, 23, 24, 71
C. Fleurent and J.A. Ferland. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 63(3):437–461, 1996.

[Fleurent and Ferland, 1996b] cited page 18, 36, 76
C. Fleurent and J.A. Ferland. Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability. In *Cliques, Coloring, and Satisfiability Second DIMACS Implementation Challenge* [Johnson and Trick, 1996], pages 619–652.

[Fotakis *et al.*, 2001] cited page 19
D.A. Fotakis, S.D. Likothanassis, and S.K. Stefanakos. An evolutionary annealing approach to graph coloring. *Proceedings of EvoWorkshops2001—Applications of Evolutionary Computing*, 2037:120–129, 2001.

[Frawley *et al.*, 1992] cited page 1
W.J. Frawley, G. Piatetsky-Shapiro, and C.J. Matheus. Knowledge discovery in databases: An overview. *AI Magazine*, 13(3):57–70, 1992.

[Freisleben and Merz, 1996] cited page 82
B. Freisleben and P. Merz. A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 616–621, 1996.

[Funabiki and Higashino, 2000] cited page 18
N. Funabiki and T. Higashino. A minimal-state processing search algorithm for graph coloring problems. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 83(7):1420–1430, 2000.

[Galinier and Hao, 1999] cited page 13, 14, 14, 14, 14, 19, 22, 23, 23, 30, 36, 67, 70, 70, 71, 73, 76, 82, 89, 91
P. Galinier and J.K. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999.

[Galinier and Hertz, 2006] cited page 12, 23
P. Galinier and A. Hertz. A survey of local search methods for graph coloring. *Computers and operations research*, 33(9):2547–2562, 2006.

[Galinier *et al.*, 2008] cited page 14, 14, 19, 22, 36, 67, 71, 86
P. Galinier, A. Hertz, and N. Zufferey. An adaptive memory algorithm for the k-coloring problem. *Discrete Applied Mathematics*, 156(2):267–279, 2008.

[Galinier, 1999] cited page 12
P. Galinier. *Etude des métahueristiques pour la satisfaction de contraintes et la coloration.* PhD thesis, Université de Montpellier, France, 1999.

[Gamache *et al.*, 2007] cited page 13
M. Gamache, A. Hertz, and J.O. Ouellet. A graph coloring model for a feasibility problem in monthly crew scheduling with preferential bidding. *Computers and Operations Research*, 34(8):2384–2395, 2007.

[Gamst and Rave, 1982] cited page 13
A. Gamst and W. Rave. On frequency assignment in mobile automatic telephone systems. In *Proceedings of the IEEE Global Communications Conference*, pages 309–315, 1982.

[Garey *et al.*, 1979] cited page 6, 6, 12
M.R. Garey, D.S. Johnson, et al. *Computers and Intractability: A Guide to the Theory of NP-completeness.* Freeman, San Francisco, U.S.A., 1979.

[Gebremedhin *et al.*, 2005] cited page 13
A.H. Gebremedhin, F. Manne, and A. Pothen. What color is your jacobian? graph coloring for computing derivatives. *SIAM Review*, 47(4):629, 2005.

[Gerber *et al.*, 1998] cited page 39
M.U. Gerber, P. Hansen, and A. Hertz. Local optima topology for the 3-sat problem. *Cahiers du GERAD*, G–98–68, 1998.

[Glass and Pruegel-Bennett, 2005] cited page 56, 56
C.A. Glass and A. Pruegel-Bennett. A polynomially searchable exponential neighbourhood for graph colouring. *Journal of the Operational Research Society*, 56(3):324–330, 2005.

[Glass, 2002] cited page 13
CA Glass. Bag rationalisation for a food manufacturer. *Journal of the Operational Research Society*, pages 544–551, 2002.

[Glover and Laguna, 1997] cited page 14, 21, 51
F. Glover and M. Laguna. *Tabu Search.* Springer, 1997.

[Glover *et al.*, 1996] cited page 19, 28
F. Glover, M. Parker, and J. Ryan. Coloring by tabu branch and bound. In *Cliques, Coloring, and Satisfiability Second DIMACS Implementation Challenge* [Johnson and Trick, 1996], pages 285–307.

[Glover, 1986] cited page 7, 14, 21
F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations research*, 13(5):533–549, 1986.

[Goldberg and Richardson, 1987] cited page 83, 83, 83
D.E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application table of contents*, pages 41–49. L. Erlbaum Associates Inc. Hillsdale, NJ, USA, 1987.

[Gusfield, 2002] cited page 56, 56, 56, 93, 95, 95
D. Gusfield. Partition-distance a problem and class of perfect graphs arising in clustering. *Information Processing Letters*, 82(3):159–164, 2002.

[Hale, 1980] cited page 13
W.K. Hale. Frequency assignment theory and applications. *Proceedings of the IEEE*, 68(12):1497–1514, 1980.

[Hamiez and Hao, 2001] cited page 19, 70, 76, 81
J. P. Hamiez and J. K. Hao. Scatter search for graph coloring. In *Artificial Evolution*, volume 2310 of *LNCS*, pages 168–179. Springer, 2001.

[Hamiez and Hao, 2004] cited page 19, 39, 40
J.P. Hamiez and J.K. Hao. An analysis of solution properties of the graph coloring problem. In *Metaheuristics computer decision-making*, pages 325–345. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[Hansen, 1986] cited page 21
P. Hansen. The steepest ascent mildest descent heuristic for combinatorial programming. *Congress on Numerical Methods in Combinatorial Optimization, Capri, Italy*, pages 70–145, 1986.

[Hao *et al.*, 1999] cited page 6
J.K. Hao, P. Galinier, and M. Habib. Metaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes. *Revue d'Intelligence Artificielle*, 13(2):283–324, 1999.

[Hertz and Werra, 1987] cited page 13, 14, 21, 23
A. Hertz and D. Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4):345–351, 1987.

[Hertz *et al.*, 1994] cited page 25, 39, 39, 40
A. Hertz, B. Jaumard, and M.P. de Aragão. Local optima topology for the k-coloring problem. *Discrete Applied Mathematics*, 49(1-3):257–280, 1994.

[Hertz *et al.*, 2008] cited page 14, 14, 14, 18, 22, 34, 36, 41, 65, 65, 65, 67, 86, 86, 87
A. Hertz, A. Plumettaz, and N. Zufferey. Variable space search for graph coloring. *Discrete Applied Mathematics*, 156(13):2551–2560, 2008.

[Hoos and Stützle, 2004] cited page 6, 7, 23, 24
H. Hoos and T. Stützle. *Stochastic Local Search Foundations & Applications.* Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2004.

[Horn *et al.*, 1994] cited page 39
J. Horn, D.E. Goldberg, and K. Deb. Long path problems. In *PPSN 94*, volume 866 of *LNCS*, pages 149–158. Springer, 1994.

[Johnson and Trick, 1996] cited page 14, 16, 112, 114, 116
D.S. Johnson and M. Trick. *Cliques, Coloring, and Satisfiability Second DIMACS Implementation Challenge*, volume 26 of *DIMACS series in Discrete Mathematics and Theoretical Computer Science.* American Mathematical Society, 1996.

[Johnson *et al.*, 1991] cited page 13, 14, 16, 16, 16, 16, 28
D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by simulated annealing an experimental evaluation; part ii, graph coloring and number partitioning. *Operations Research*, 39(3):378–406, 1991.

[Johnson *et al.*, 2002] cited page 14
D. Johnson, A. Mehrotra, and M. Trick. Computational Symposium on Graph Coloring and Its Generalizations, COLOR02, Cornell University, Ithaca, NY, 2002. (`mat.gsia.cmu.edu/COLORING02/`).

[Johnson *et al.*, 2008] cited page 12, 14
D.S. Johnson, A. Mehrotra, and M.A. Trick. Special issue on computational methods for graph coloring and its generalizations. *Discrete Applied Mathematics*, 156(2):145–146, 2008.

[Jones and Forrest, 1995] cited page 39, 39
T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 184–192, 1995.

[Kallel *et al.*, 2001] cited page 39
L. Kallel, B. Naudts, and C. R. Reeves. Properties of fitness functions and search landscapes. In *Theoretical aspects of evolutionary computing*, pages 175–206. Springer-Verlag, London, UK, 2001.

[Karp, 1972] cited page 12
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, 1972.

[Kauffman and Levin, 1987] cited page 39
S. Kauffman and S. Levin. Towards a general theory of adaptive walks on rugged landscapes. *Journal of theoretical Biology*, 128(1):11, 1987.

[Kendall, 1938] cited page 67
MG Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

[Konovalov *et al.*, 2005] cited page 93, 94, 95
D.A. Konovalov, B. Litow, and N. Bajema. Partition-distance via the assignment problem. *Bioinformatics*, 21(10):2463–2468, 2005.

[Kruskal, 1964] cited page 43, 43
J.B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

[Kühn, 1955] cited page 93, 95
H.W. Kühn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[Kuntz *et al.*, 2004] cited page 39
P. Kuntz, B. Pinaud, and R. Lehn. Elements for the description of fitness landscapes associated with local operators for layered drawings of directed graphs. In *Metaheuristics Computer Decision-Making*, pages 405–420. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[Laurent and Hao, 2009] cited page 13
B. Laurent and J.K. Hao. List-graph colouring for multiple depot vehicle scheduling. *International Journal of Mathematics in Operational Research*, 1(1):228–245, 2009.

[Leighton, 1979] cited page 13, 16, 64
F.T. Leighton. A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, 84(6):489–503, 1979.

[Lerman *et al.*, 1981] cited page 7
I.C. Lerman, T. Chantrel, and I. Cohen. *Classification et analyse ordinale des données.* Dunod Paris, 1981.

[Lewandowski and Condon, 1996] cited page 13, 13, 16, 17
G. Lewandowski and A. Condon. Experiments with parallel graph coloring heuristics and applications of graph coloring. In *Cliques, Coloring, and Satisfiability Second DIMACS Implementation Challenge* [Johnson and Trick, 1996], pages 309–334.

[Lim and Wang, 2005] cited page 13
A. Lim and F. Wang. Robust graph coloring for uncertain supply chain management. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05)-Track 3-Volume 03*. IEEE Computer Society, 2005.

[Lü and Hao, 2009] cited page 14, 19
Z. Lü and J.K. Hao. A critical element-guided perturbation strategy for iterated local search. In *EvoCOP*, volume 5482 of *LNCS*, pages 1–12, 2009.

[Lü and Hao, 2010] cited page 13, 19, 36, 67, 88, 91, 91
Z. Lü and J.K. Hao. A memetic algorithm for graph coloring. *European Journal of Operational Research*, 203(2):241–250, 2010.

[Lucet *et al.*, 2006] cited page 13
C. Lucet, F. Mendes, and A. Moukrim. An exact method for graph coloring. *Computers and Operations Research*, 33(8):2189–2207, 2006.

[Lund and Yannakakis, 1994] cited page 12
C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981, 1994.

[Mahfoud, 1995a] cited page 83, 83, 83
S.W. Mahfoud. A comparison of parallel and sequential niching methods. In L. J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 136–143, 1995.

[Mahfoud, 1995b] cited page 83, 83
S.W. Mahfoud. *Niching methods for genetic algorithms.* PhD thesis, University of Illinois at Urbana-Champaign, 1995.

[Malaguti and Toth, 2008] cited page 14, 70, 76, 76
E. Malaguti and P. Toth. An evolutionary approach for bandwidth multicoloring problems. *European Journal of Operational Research*, 189(3):638–651, 2008.

[Malaguti and Toth, in press] cited page 6, 12, 13
E. Malaguti and P. Toth. A survey on vertex coloring problems. *International Transactions in Operational Research*, in press. `http://dx.doi.org/10.1111/j.1475-3995.2009.00696.x`.

[Malaguti *et al.*, 2008] cited page 13, 14, 14, 19, 22, 28, 34, 36, 36, 67, 70, 71, 86, 87, 91
E. Malaguti, M. Monaci, and P. Toth. A metaheuristic approach for the vertex coloring problem. *INFORMS Journal on Computing*, 20(2):302, 2008.

[Marino *et al.*, 1999] cited page 76
A. Marino, A. Prugel-Bennett, and C.A. Glass. Improving graph colouring with linear programming and genetic algorithms. In *Proceedings of Eurogen99*, pages 113–118. University of Jyväskylä, Finland, 1999.

[Maturana, 2009] cited page 9
J. Maturana. *Contrôle générique de Paramètres pour les Algorithmes Évolutionnaires.* PhD thesis, University of Angers, France, 2009.

[Mehrotra and Trick, 1996] cited page 13
A. Mehrotra and M.A. Trick. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8(4):344–354, 1996.

[Méndez-Díaz and Zabala, 2006] cited page 13, 13
I. Méndez-Díaz and P. Zabala. A branch-and-cut algorithm for graph coloring. *Discrete Applied Mathematics*, 154(5):826–847, 2006.

[Merz and Freisleben, 2000a] cited page 39
P. Merz and B. Freisleben. Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Transactions on Evolutionary Computation*, 4(4):337–352, 2000.

[Merz and Freisleben, 2000b] cited page 39
P. Merz and B. Freisleben. Fitness landscapes, memetic algorithms, and greedy operators for graph bipartitioning. *Evolutionary Computation*, 8(1):61–91, 2000.

[Merz, 2004] cited page 39, 39, 83
P. Merz. Advanced fitness landscape analysis and the performance of memetic algorithms. *Evolutionary Computation*, 12(3):303–325, 2004.

[Miller and Shaw, 1996] cited page 83, 83, 83, 83, 83, 83
B. L. Miller and M.J. Shaw. Genetic algorithms with dynamic niche sharing for multimodalfunction optimization. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 786–791, 1996.

[Morgenstern, 1996] cited page 14, 14, 18, 28, 34, 36, 67, 71, 87, 91
C. Morgenstern. Distributed coloration neighborhood search. In *Cliques, Coloring, and Satisfiability Second DIMACS Implementation Challenge* [Johnson and Trick, 1996], pages 335–358.

[Paquete and Stützle, 2002] cited page 14, 18, 36
L. Paquete and T. Stützle. An experimental investigation of iterated local search for coloring graphs. In S. Cagnoni et al., editors, *EvoWorkshops*, volume 2279 of *LNCS*, pages 121–130. Springer, 2002.

[Park and Lee, 1996] cited page 13
T. Park and C.Y. Lee. Application of the graph coloring algorithm to the frequency assignment problem. *Journal of the Operations Research Society of Japan*, 39(2):258–265, 1996.

[Porumbel *et al.*, 2007a] cited page 3, 21
C.D. Porumbel, J.K. Hao, and P. Kuntz. Reinforced tabu search for graph coloring. *submitted paper available on request*, 2007. revised in 2009.

[Porumbel *et al.*, 2007b] cited page 3
C.D. Porumbel, J.K. Hao, and P. Kuntz. A study of evaluation functions for the graph k-coloring problem. In *EA07*, volume 4926 of *LNCS*, pages 124–135, 2007.

[Porumbel *et al.*, 2008] cited page 3, 92
C.D. Porumbel, J.K. Hao, and P. Kuntz. An efficient algorithm for computing the partition distance. *submitted paper available on request*, 2008. revised in 2009.

[Porumbel *et al.*, 2009a] cited page 3, 69
C.D. Porumbel, J.K. Hao, and P. Kuntz. Diversity control and multi-parent recombination for evolutionary graph coloring algorithms. In *EvoCOP*, volume 5482 of *LNCS*, pages 121–132, 2009.

[Porumbel *et al.*, 2009b] cited page 3
C.D. Porumbel, J.K. Hao, and P. Kuntz. An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. *submitted paper available on request*, 2009.

[Porumbel *et al.*, 2009c] cited page 2, 38
C.D. Porumbel, J.K. Hao, and P. Kuntz. Position guided tabu search for graph coloring. In *LION (Learning in OptimizatioN)*, submitted to LNCS, 2009.

[Porumbel *et al.*, 2010] cited page 2, 38, 49, 67, 67, 87, 91
C.D. Porumbel, J.K. Hao, and P. Kuntz. A search space "cartography" for guiding graph coloring heuristics. *Computers & Operations Research*, 37:769–778, 2010.

[Prestwich, 2002] cited page 19, 87
S. Prestwich. Coloration neighbourhood search with forward checking. *Annals of Mathematics and Artificial Intelligence*, 34(4):327–340, 2002.

[Radcliffe, 1994] cited page 70
N.J. Radcliffe. The algebra of genetic algorithms. *Annals of Mathematics and Artificial Intelligence*, 10(4):339–384, 1994.

[Rajagopalan *et al.*, 2008] cited page 82
R. Rajagopalan, C. K. Mohan, K. Mehrotra, and P. K. Varshney. EMOCA An Evolutionary multi-objective crowding algorithm. *Journal of Intelligent Systems*, 17(1/3):107, 2008.

[Ramani *et al.*, 2006] cited page 13
A. Ramani, I.L. Markov, K.A. Sakallah, and F.A. Aloul. Breaking instance-independent symmetries in exact graph coloring. *Journal of Artificial Intelligence Research*, 26:191–224, 2006.

[Reeves and Yamada, 1998] cited page 39
C.R. Reeves and T. Yamada. Genetic algorithms, path relinking, and the flowshop sequencing problem. *Evolutionary Computation*, 6(1):45–60, 1998.

[Reeves, 1997] cited page 70
C.R. Reeves. Genetic algorithms for the operations researcher. *INFORMS Journal on Computing*, 9(3):231–250, 1997.

[Régnier, 1983] cited page 93
S. Régnier. Sur quelques aspects mathématiques des problèmes de classification automatique. *Mathématiques et Sciences Humaines*, 82:20, 1983. reprint of *ICC Bulletin*, 4, 175- 191, Rome, 1965.

[Rodriguez-Tello *et al.*, 2008a] cited page 36
E. Rodriguez-Tello, J.K. Hao, and J. Torres-Jimenez. An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem. *Computers & Operations Research*, 35(10):3331–3346, 2008.

[Rodriguez-Tello *et al.*, 2008b] cited page 36
E. Rodriguez-Tello, J.K. Hao, and J. Torres-Jimenez. An improved simulated annealing algorithm for bandwidth minimization. *European Journal of Operational Research*, 185(3):1319–1335, 2008.

[Russell and Norvig, 2002] cited page 1
S.J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, New Jersey, U.S.A., 2002.

[Ryan, 1995] cited page 39
J. Ryan. The depth and width of local minima in discrete solution spaces. *Discrete Applied Mathematics*, 56(1):75–82, 1995.

[Sewell, 1996] cited page 13
E.C. Sewell. An improved algorithm for exact graph coloring. In *Cliques, Coloring, and Satisfiability Second DIMACS Implementation Challenge* [Johnson and Trick, 1996], pages 359–376.

[Shimodaira, 1997] cited page 82
H. Shimodaira. Dcga a diversity control oriented genetic algorithm. In *Proceedings of the 9th IEEE International Conference on Tools with Artificial Intelligence*, pages 367–374, 1997.

[Smith *et al.*, 1993] cited page 83, 83, 83, 83, 83
R.E. Smith, S. Forrest, and A.S. Perelson. Searching for diverse, cooperative populations with genetic algorithms. *Evolutionary Computation*, 1(2):127–149, 1993.

[Sörensen and Sevaux, 2006] cited page 82
K. Sörensen and M. Sevaux. Ma|pm memetic algorithms with population management. *Computers and Operations Research*, 33(5):1214–1225, 2006.

[Stadler and Schnabl, 1992] cited page 39
P.F. Stadler and W. Schnabl. The landscape of the traveling salesman problem. *Physical Letters A*, 161(4):337–344, 1992.

[Streeter and Smith, 2006] cited page 39, 39, 39
M.J. Streeter and S.F. Smith. How the landscape of random job shop scheduling instances depends on the ratio of jobs to machines. *Journal of Artificial Intelligence Research*, 26:247–287, 2006.

[Tagawa *et al.*, 1999] cited page 76
K. Tagawa, K. Kanesige, K. Inoue, and H. Haneda. Distance based hybrid genetic algorithm: an application for the graph coloring problem. In *Proceedings of the 1999 Congress on Evolutionary Computation*, pages 2325–2332, 1999.

[Talbi and Weinberg, 2007] cited page 93
E.G. Talbi and B. Weinberg. Breaking the search space symmetry in partitioning problems an application to the graph coloring problem. *Theoretical Computer Science*, 378(1):78–86, 2007.

[Tomassini *et al.*, 2008] cited page 39
M. Tomassini, S. Verel, and G. Ochoa. Complex-network analysis of combinatorial spaces: The nk landscape case. *Physical Review E*, 78(6):66114, 2008.

[Trick and Yildiz, 2007] cited page 14
M. A. Trick and H. Yildiz. A large neighborhood search heuristic for graph coloring. In *CPAIOR 2007*, volume 4510 of *LNCS*, pages 346–360. Springer, 2007.

[Ursem, 2002] cited page 82, 82, 82
R. K Ursem. Diversity-guided evolutionary algorithms. In *PPSN VII*, volume 2439 of *LNCS*, pages 462–471. Springer, 2002.

[Voudouris and Tsang, 2003] cited page 8
C. Voudouris and E.P.K. Tsang. Guided local search. In F. Glover et al., editors, *Handbook of metaheuristics*, pages 185–218. Kluwer Academic Publishers, 2003.

[Weinberg, 2004] cited page 12
B. Weinberg. *Analyse et Résolution Approchées de Problèmes d'Optimisation Combinatoire: Application au Problème de Coloration de Graphe.* PhD thesis, Université de Lille 1, France, 2004.

[Welsh and Powell, 1967] cited page 13
D.J.A. Welsh and M.B. Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, 10(1):85–86, 1967.

[Woo *et al.*, 1991] cited page 13
T.K. Woo, SYW Su, and R. Newman-Wolfe. Resource allocation in a dynamically partitionable bus network using a graph coloring algorithm. *IEEE Transactions on Communications*, 39(12):1794–1801, 1991.

[Zdeborová and Krzakała, 2007] cited page 44, 44
L. Zdeborová and F. Krzakała. Phase transitions in the coloring of random graphs. *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics)*, 76(3):031131, 2007.

[Zhang, 2004] cited page 39
W. Zhang. Configuration landscape analysis and backbone guided local search. part i satisfiability and maximum satisfiability. *Artificial Intelligence*, 158(1):1–26, 2004.

[Zhu, 2003] cited page 82, 82, 82
KQ Zhu. A diversity-controlling adaptive genetic algorithm for the vehicle routing problem with time windows. In *Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence*, pages 176–183, 2003.

[Zufferey *et al.*, 2008] cited page 13
N. Zufferey, P. Amstutz, and P. Giaccari. Graph colouring approaches for a satellite range scheduling problem. *Journal of Scheduling*, 11(4):263–277, 2008.

[Zufferey, 2002] cited page 12
N. Zufferey. *Heuristiques pour les problèmes de coloration des sommets d'un graphe et d'affectation de fréquences avec polarités.* PhD thesis, École Polytechnique Fédérale de Lausanne, 2002.

Publications

International Journals

- Daniel Porumbel, Jin-Kao Hao and Pascale Kuntz, A Search Space "Cartography" for Guiding Graph Coloring Heuristics. *Computers & Operations Research*, vol. 37 (2010), pp. 769-778.

International Journals (submitted or in revision)

- Daniel Porumbel, Jin-Kao Hao and Pascale Kuntz, Reinforced Tabu Search for Graph Coloring.

- Daniel Porumbel, Jin-Kao Hao and Pascale Kuntz, An Efficient Algorithm for Computing the Partition Distance.

- Daniel Porumbel, Jin-Kao Hao and Pascale Kuntz, An Evolutionary Approach with Diversity Guarantee and Well-Informed Grouping Recombination for Graph Coloring.

International Conferences

- Daniel Porumbel, Jin-Kao Hao and Pascale Kuntz, Diversity Control and Multi-Parent Recombination for Evolutionary Graph Coloring Algorithms. Proceedings of Evocop 2009 (*9th European Conference on Evolutionary Computation in Combinatorial Optimisation*, Tübingen, Germany), LNCS 5482: 121-132, Springer, 2009. [selected among the three best paper nominees]

- Daniel Porumbel, Jin-Kao Hao and Pascale Kuntz, Position Guided Tabu Search for Graph Coloring. Oral presentation at the LION 3 (*Learning and Intelligent OptimizatioN* conference, Trento, Italy), to appear in LNCS post-proceedings, Springer, 2009.

- Daniel Porumbel, Jin-Kao Hao and Pascale Kuntz, A study of evaluation functions for the graph K-coloring problem. Selected papers from EA'07 (*8th International Conference on Artificial Evolution*, Tours, France), LNCS 4926: 124-135, Springer, 2007.

Other Conferences and Talks

- Daniel Porumbel, Jin-Kao Hao and Pascale Kuntz, Guided Local Search for Graph Coloring (in French). Presented at the 10th congress of the French Operational Research Society. Nancy, France, 2009.
- Daniel Porumbel, Jin-Kao Hao and Pascale Kuntz, A Guided Algorithm for Graph Coloring (in French). Talk at the *Graphes and Algorithms* workshop. Sophia-Antipolis, Nice, France 2009.
- Daniel Porumbel, Jin-Kao Hao and Pascale Kuntz, Reinforced Tabu Search for Graph Coloring, Presented at the 9th congress of the French Operational Research Society Clermont-Ferrand, France, 2008.

Invited Talk

- Daniel Porumbel, Tabu Search for Graph Coloring, Reinforced Tabu Search for Graph Coloring, Talk at the ASAP (www.asap.cs.nott.ac.uk) research team seminar, University of Nottingham, United Kingdom, 2008

Heuristic Algorithms And Learning Techniques – Applications to the Graph Coloring Problem –

Abstract

The last couple of decades have seen a surge of interest and sophistication in using heuristics to solve combinatorial optimization problems. However, the theoretical and practical research of these algorithms show there are many important challenges yet to be overcome – e.g. it seems that it is quite difficult to make a heuristic integrate a global vision over its own evolution or over its trajectory. An important risk is to loop repeatedly between a *limited* number of different local optima and to explore only few regions even by using a very long time. Taking the well-known graph coloring problem as an experimental framework, we develop several new heuristics that integrate certain learning mechanisms so as to render the search process more “self-aware”. For instance, we introduce an algorithm that is able to record its trajectory and to interpret its own evolution. A search space analysis showed that the best discovered potential solutions tend to occur relatively close to each other, clustered in spheres of fixed radius. Using such learned information, we developed: (i) diversification algorithms that ”pay attention” not to visit the same sphere repeatedly, (ii) intensification algorithms assuring an in-depth exploration of a closed perimeter using a breath-first-search traversal of its spheres, or (iii) evolutionary algorithms that are able to keep the individuals sufficiently distant at all times, while not sacrificing population quality. In fact, we present numerous other techniques (e.g. new evaluation functions) that are able to render the heuristic search more “well-informed”.

Keywords : search while learning, graph coloring, exploration path recording, intensification heuristics, population diversity, search space distance

Algorithmes Heuristiques et Techniques d'Apprentissage – Applications au Problème de Coloration de Graphe –

Résumé

Au cours des trois dernières décennies, les algorithmes heuristiques ont permis de réaliser des progrès remarquables dans la résolution des problèmes difficiles d'optimisation combinatoire. Cependant, la conception de ces algorithmes relève encore plusieurs challenges importants – en particulier, il semble qu'il est toujours difficile d'intégrer dans une heuristique une vue d'ensemble sur l'évolution de la recherche ou sur sa trajectoire. Prenant comme cadre expérimental le problème bien connu de la coloration de graphe, nous présentons de nouvelles stratégies qui font appel à certains mécanismes d'apprentissage pour rendre le processus de recherche plus “auto-conscient”. Nous introduisons un algorithme qui est capable d'enregistrer sa trajectoire et d'interpréter sa propre évolution. Une analyse de l'espace de recherche a montré que les meilleures configurations visitées sont relativement proches les unes des autres, regroupées dans des *sphères* de rayon fixe. Avec ce type d'informations apprises, nous avons conçu : (i) des algorithmes de diversification qui “prennent garde” à ne pas visiter la même sphère à plusieurs reprises, (ii) des algorithmes d'intensification qui se focalisent sur l'exploration d'un périmètre limité en utilisant un parcours en largeur des sphères de ce périmètre, et (iii) des approches évolutionnistes pour gérer la diversité de sorte que les individus soient à la fois de bonne qualité eu égard à la fonction objectif et suffisamment distants les uns des autres. En fait, nous présentons une gamme de techniques (e.g. nouvelles fonctions d'évaluation) qui peuvent rendre la recherche heuristique “bien informée”.

Mots clés : apprentissage et optimisation, enregistrement de l'évolution de la recherche, heuristique d'intensification, diversité de la population, distance dans l'espace de recherche