Adaptive Memory Programming:
An Empirical Study with the Bandwidth Coloring Problem

RAFAEL MARTÍ
Departamento de Estadística e Investigación Operativa
Universitat de València, Spain
rafael.marti@uv.es

FRANCISCO GORTAZAR
Departamento de Ciencias de la Computación
Universidad Rey Juan Carlos, Spain
francisco.gortazar@urjc.es

ABRAHAM DUARTE
Departamento de Ciencias de la Computación
Universidad Rey Juan Carlos, Spain
abraham.duarte@urjc.es

Version: May 7, 2009

ABSTRACT

In this article we study the use of memory structures in both constructive and improvement methods. Typical tabu search applications employ short term memory structures to overcome local optimality. However, other memory structures, such as frequency-based constructions have often been ignored or implemented in naïve ways that disregard important elements of the original proposals. In this paper we describe the adaptation of these memory programming elements to the bandwidth coloring problem. We also propose constructive and improvement methods based on GRASP, a well-known memory-less methodology, to compare and hybridize both designs.

The bandwidth coloring problem consists of assigning a color to each vertex of a graph, so that the absolute value of the difference between the colors of adjacent vertices is at least the value of the weight of the associated edge. This problem generalizes the classical vertex coloring problem and different heuristics have recently been proposed to obtain high quality solutions. Comparison of our results with previously reported instances and existing heuristics indicate that the methods we propose are competitive and require short computational times. Our findings also disclose that memory appears to play a more important role during improvement phases of search than during constructive phases.

Key Words: Graph coloring, Tabu Search, GRASP.

1. Introduction

From a naive standpoint, virtually all heuristics other than complete randomization induce a pattern whose present state depends on the sequence of past states, and therefore incorporate an implicit form of "memory." However, such an implicit memory, as indicated in Glover and Laguna (1997), does not take a form normally viewed to be a memory structure. By contrast, the explicit use of memory structures constitutes the core of a large number of intelligent solving methods. They include tabu search, scatter search and evolutionary path relinking among others. These methods focus on exploiting a set of strategic memory designs. Tabu search (TS), the metaheuristic that launched this perspective, is the source of the term Adaptive Memory Programming (AMP) to describe methods that use advanced memory strategies (and hence learning, in a non-trivial sense) to guide a search. (See, e.g., Glover, 1996.) In linguistic terms, to define semantic hierarchies, we can say that AMP is the hyperonym of tabu search in a similar way that mathematical programming is the hyperonym of linear programming.

Over time, various uses of memory strategies have also become incorporated into a variety of other metaheuristics. For example, a number of "hybrid" genetic and evolutionary methods have arisen that embed some form of these memory strategies within them, and more recently several have appeared that have dropped the hybrid nomenclature (and in some cases reference to tabu search). Today it is not unusual for evolutionary methods to implement long term memory structures to record elite solutions found during the search for intensification or diversification purposes. In this paper we study the inclusion of memory structures in combined construction-improvement methods, comparing memory-based with memory-less designs.

To set the stage for discussing the strategies we investigate, it is useful to briefly sketch some of the features of tabu search. TS is a metaheuristic that guides a local search heuristic procedure to explore the solution space beyond local optimality. Its use of adaptive memory and associated strategies for exploiting such memory, creates a flexible search behavior and offers a means to learn improved trajectories through the solution space. The structure of a neighborhood in tabu search goes beyond that typically employed in local search by embracing the types of moves used in constructive and destructive processes (where the foundations for such moves are accordingly called *constructive* and *destructive neighborhoods*). As described in Glover (1996), adaptive memory in these settings involves an attribute-based focus and closely depends on the elements of *recency*, *frequency* and *influence*. In this paper we explore the adaptation of frequency memory structures in the context of constructive and local search methods.

We can implement memory structures within a constructive process to favor (or avoid) the inclusion of certain elements in a solution previously identified as attractive (or unattractive). Such expanded uses of the neighborhood concept reinforce a fundamental perspective of TS, which is to define neighborhoods in dynamic ways that can include serial or simultaneous consideration of multiple types of moves. Constructive neighborhoods have been proposed from the very beginning of the methodology, as documented in Glover and Laguna (1997); however, they have rarely been applied in TS implementations (Duarte and Martí, 2007). In this paper we explore the adaptation of memory structures, in both constructive and improvement methods, to a hard

combinatorial optimization problem: the bandwidth coloring problem. We compare this memory-based design with a memory-less design based on the GRASP methodology. We will see that within the constructive phase both approaches, memory-based and memory-less, can be effective, and hence advantages derived from the use of memory-based strategies appear to rest primarily in the improvement phase. (as discussed in the Conclusion.) Independent of this finding, our methods achieve a performance ranking that shows them to be competitive with leading methods recently proposed to solve the bandwidth coloring problem.

A simple first approach to measuring the global impact of a methodology could be to perform a search over the Internet. We have used *Google*, a well-known search engine, and made the query "metaheuristics", obtaining 165,000 entries. We have then refined the search with the query "GRASP", obtaining 10,700 entries. Alternatively, if we refine the search with the query "memory" we obtain 33,900 entries (the same as if we search for "tabu search"). Note that the number of entries found may change due to multiple factors (such as the search engine, the specific term that we are searching for or even the country or the day). It is clear that we cannot examine all these entries in practice, but at first sight we realized that most of the "memory" entries focus on local search based methods and pay limited attention to the construction process. On the other hand, the "GRASP" papers linked to the associated entries mainly describe a construction process and usually resort to a standard local search method without considering possible hybridizations or refinements (like the inclusion of short-term memory structures). The objective of this paper is to illustrate that memory-based constructions can be as effective as memory-less constructions (that typically appear in GRASP), and, on the other hand, that the addition of simple memory structures to local search methods (short-term components) can improve their performance (even in limited running times) and therefore, we can conclude that hybrid GRASP procedures, in which simple memory structures are added to the typical designs, may be a good choice when designing a solving method.

2. The Case Problem

Let $G=(V,E)$ be a graph with a vertex set V ($|V| = n$) and an edge set E ($|E| = m$) with a strictly positive integer weight d_{ij} associated to each edge $(i, j) \in E$. A k-coloring c of G labels each vertex $i \in V$ with an integer $c(i) \in \{1, 2, \ldots, k\}$ (called color) in such a way that $|c(i) - c(j)| \geq d_{ij}$ for all $(i, j) \in E$. The bandwidth coloring problem (BCP) consists of finding a k coloring with the smallest value of k. In mathematical terms:

$$\text{(BCP) Minimize } k$$

$$\text{s.t.:} \quad |c(i) - c(j)| \geq d_{ij} \qquad \forall (i, j) \in E$$

$$c(i) \in \{1, 2, \ldots, k\} \; \forall i \in E$$

The classical vertex coloring problem (VCP) is a particular case of the BCP in which $d_{ij}=1$ for all $(i, j) \in E$. The BCP is therefore NP-hard since it generalizes the VCP (Garey and Johnson 1979). The bandwidth coloring problem, as well as other generalizations of the classical vertex coloring problem (such as the multi-coloring or the T-coloring problem), allows complex real problems to be modeled, like for example the assignment of frequencies to different cells in a mobile network (Malaguti and Toth

2008). Specifically, the allocation of frequencies to transmitters to avoid interferences above a given threshold triggered the interest on these variants of the coloring problem. In this paper we restrict our attention to the BCP. The classical coloring problem has been extensively studied during the last two decades and a large number of papers are devoted to it. However, the BCP has received much less attention, mostly dating from the *Computational Symposium on Graph Coloring and its Generalizations* in 2002.

Phan and Skiena (2002) proposed a context-independent method instantiated for coloring problems. Their method, called Discropt, is a black-box solver for problems where variables take an integer value within a range. Prestwich (2002) extended the FCNS algorithm originally proposed for VCP to the BCP. The method basically combines a local search with a constraint propagation algorithm. He introduced the *domain* as the number of possible colors for a vertex. The constructive method considers the domain cardinality as the measure to sort the vertices for selection in the coloring process. The computational results show that this method is able to obtain the best solutions in short computational times.

The simplest heuristic methods for the VCP are the *sequential coloring* approaches. First, the vertices are sorted, and the top vertex is labeled (colored) with number one. The remaining vertices are considered in order, and each vertex is labeled with the first color for which it has no adjacent vertices already labeled with this color. Several different schemes have been used for the initial ordering. The Largest First (LF) approach of Welsh and Powell (1967) sorts the vertices in decreasing degree. Although these methods are easy to implement and fast, they often produce colorings which are far from optimal. Lim et al. (2005) proposed a multi-start method for the BCP in which the initial solution is obtained with a greedy method, SEQ, which basically adapts the LF approach. Given an ordering of vertices, their greedy algorithm sequentially assigns the smallest color to each vertex verifying the *bandwidth constraints* ($|c(i) - c(j)| \geq d_{ij}$ for all $(i, j) \in E$). At each iteration, the method first generates a sequence of nodes and then produces a solution of the BCP with the greedy algorithm. Then, it tries to improve this solution by reducing the number of colors used by one unit below the number of colors in the best solution known so far. The authors applied their method to geometric graphs (Geom) and obtained similar results to the method by Prestwich (2002).

Malaguti and Toth (2008) proposed a combination of evolutionary and tabu search methodologies. As in the method by Lim et al. (2005) the algorithm first constructs an initial solution with a sequential method and then tries to improve it by reducing the number of colors used in the constructed solution by one unit. The sequential method is an adaptation of the well-known DSATUR algorithm (Brèlaz, 1979) for the VCP in which vertices are selected at each stage based on its *score* or *saturation degree* — the number of distinctly colored adjacent vertices. A vertex with the maximum saturation degree is selected and labeled with the first legal color. The authors introduced a new score $s(i)$ of vertex i as the sum of the maximum distances between i and each adjacent color.

$$s(i) = \sum_{h=1}^{k} \max \left\{ d_{ij} : j \in N(i), c(j) = h \right\}$$

where $N(i)$ is the set of vertices adjacent to i and k is the maximum color currently used. This score reflects the urgency to color a vertex. Therefore vertices with high score are colored first. The improvement method applies a tabu search algorithm, T1, to the constructed solution in which those vertices with maximum color k are uncolored. The method tries to color the uncolored vertices with an integer in {1, 2, ..., k-1}, thus improving it. If the method succeeds, k is decreased to k-1 and the process is iterated. The short-term memory structure prevents a vertex from taking the same color it took in the previous iterations.

The construction (DSATUR) and tabu search (T1) procedures are embedded in an evolutionary algorithm for improved outcomes. The population is created by successively applying the greedy sequential method. A crossover operator combines randomly selected solutions and the tabu search method is applied to improve the offspring. The improved solution replaces the worst parent in the population. This hybrid method obtains the best known solutions for this problem; however its running times are extremely time-consuming, several orders of magnitude larger than the other methods based on the combination of construction and improvement algorithms (that can be estimated in several hours of CPU time). We will then consider in our computational comparison the propagation method by Prestwich (2002), FCNS, the constructive with improvement method by Lim et al. (2005), Multistart, and the construction with tabu search by Malaguti and Toth (2008), DSATUR+T1.

3. Constructive Methods

In this section we propose two different approaches to construct good solutions for the BCP. In the first one (Section 3.1), we develop three constructive methods based on the GRASP methodology. They do not implement any memory strategy but, conversely, they are based on an independent random sampling of the solution space. In our second approach (Section 3.2), we propose two methods based on memory structures and they are guided by deterministic strategies. Previous constructive heuristics for the Bandwidth Coloring Problem are the SEQ (Lim et al. 2005) and DSATUR (Malaguti and Toth 2008). We will compare our proposals with the aforementioned methods in our computational experiments in Section 5.

Generally speaking, in each iteration a constructive method adds an element to the partial solution under construction until all the elements have been added and the solution is completed. It is based on an evaluation function to measure the attractiveness of the elements to be added. A *greedy* constructive method selects the best evaluated element at each iteration. Alternatively, a *randomized* constructive method combines the evaluation with random selections in such a way that the constructive method can be applied several times, obtaining different solutions. The GRASP methodology specifies a way to combine the evaluation and randomization elements in the construction process. On the other hand, *memory-based* constructive methods modify the evaluation function to incorporate the information recorded during past constructions to guide the process. They usually do not consider randomization but on the contrary are based on strategic and deterministic designs. In Duarte and Martí (2007) different tabu search and GRASP constructions are proposed in the context of the maximum diversity problem. Their experimental study shows that tabu search constructions compare favorably with GRASP constructions for that problem.

3.1 GRASP Constructive Methods

GRASP, Greedy Randomized Adaptive Search Procedure, is a multi-start or iterative process in which each iteration consists of two phases: construction and local search. The construction phase builds a feasible solution, whose neighborhood is explored until a local optimum is found after the application of the local search phase (Resende and Ribeiro, 2003). At each iteration of the construction phase, GRASP maintains a set of candidate elements CL that can be feasibly added to the partial solution under construction. Every candidate element i is evaluated according to a greedy function, *eval*(*i*), in order to select the next element to be added to the construction. A restricted candidate list (RCL) is created with the best elements in CL. This is the greedy aspect of the method. The element to be added to the partial solution is randomly selected from those in the RCL. This is the probabilistic aspect of the heuristic. Once the selected element is added to the partial solution, the candidate list CL is updated and its elements evaluated. This is the adaptive aspect of the heuristic. Figure 1 shows the pseudo-code of this GRASP construction, in which *C* is the set of colored vertices (initially empty) and *U* the set of uncolored vertices (initially equal to *V*).

1. Initially $C = \emptyset$, $U=V$ and $k=1$.
2. Select a vertex *i* randomly from *U*.
3. Assign the color 1 to *i* ($c(i)=1$). $C = \{i\}$, $U = U - \{i\}$

WHILE ($U \neq \emptyset$)

4. CL = *U*
5. Compute *eval*(*i*) for all *i* in CL
6. Construct RCL= { $i \in$ CL / *eval*(*i*) $\leq$ *eth* }
7. Select a vertex *i** randomly in RCL
8. Let *h* be the first (minimum) legal color for *i**
9. Label vertex *i** with color *h*: $c(i^*)=h$
10. If ($h = k+1$) do $k=k+1$.
11. $U = U - \{j\}$, $C = C \cup \{u\}$

ENDWHILE

Figure 1. Constructive method C1

In the GRASP construction above, the parameter *eth* represents a threshold on the quality of the elements. Specifically, the elements in CL with an evaluation lower than *eth* are admitted, to becoming part of RCL. This search parameter is computed as a percentage α of the rank of *eval* in CL:

$$eth = min_{i \in CL}\ eval(i) + \ \alpha\ (\ max_{i \in CL}\ eval(i) - \ min_{i \in CL}\ eval(i)\ )$$

Note that if α is equal to 0, then *eth* takes the minimum value in CL and the GRASP is a greedy construction. On the other hand, if α is equal to 1, then RCL=CL and the GRASP construction is equivalent to a random method. As recommended in Resende et al. (2009), α is randomly selected in [0, 1] for each GRASP construction. Note that in Step 8 of the algorithm in Figure 1, *h* is the first (minimum) color for the selected vertex *i** satisfying the distance constraints with its adjacent colored vertices. In mathematical terms, *h* is computed as:

$$h = \min \left\{ c \in \mathbb{Z} : \ |c - c(j)| \geq d_{ij} \ \ \forall j \in N(i^*) \cap C \right\}$$

In the above adaptation of the GRASP construction to the bandwidth coloring problem, we consider two different approaches C1 and C2. Both follow the algorithm shown in Figure 1 but differ in the way they compute the evaluation function. The evaluation in C1 is given by the minimum admissible color (considering the distance constraints with the already colored vertices) while in C2 it is based on the score introduced by Malaguti and Toth (2008):

$$\text{C1: } eval(i) = \min\left\{c \in \mathbb{Z} : |c - c(j)| \geq d_{ij} \ \ \forall j \in N(i) \cap C\right\}$$
$$\text{C2: } eval(i) = \frac{1}{s(i)}$$

C1 and C2 are basically randomized sequential constructions in which we first choose a vertex (in a greedy randomized fashion) and then assign an appropriate color. We now propose a different approach that first completes a color before introducing a new one. In the GRASP algorithm C3 outlined in Figure 2 the candidate list CL (Step 4) is formed with the uncolored vertices that can be colored with color *k*. If there is no vertex that can be colored with *k*, we make *k*=*k*+1 in Step 5 and perform a new iteration; otherwise, the method proceeds in a similar way to the previous methods computing the restricted candidate list. However, the evaluation is now computed as the number of uncolored adjacent nodes (in the first iteration it is simply the adjacent nodes: *eval*(*i*)=|*N*(*i*)|). This method adapts the GRASP construction for the VCP proposed by Laguna and Martí (2001).

1. Initially $C = \emptyset$, $U=V$ and k=1.
2. Select a vertex *i* randomly from *U*.
3. Assign the color 1 to *i* ($c(i)$=1). $C = \{i\}$, $U = U - \{i\}$

WHILE $(U \neq \emptyset)$

4. $CL = U \cap \{i \in V : |k - c(j)| \geq d_{ij} \ \ \forall j \in N(i) \cap C\}$

IF ($CL = \emptyset$)

5. $k = k + 1$

ELSE

6. Compute $eval(i) = |N(i) \cap U|$ for all *i* in CL
7. Construct RCL= { $i \in$ CL / $eval(i) \leq eth$ }
8. Select a vertex *j* randomly in RCL
9. Label vertex *j* with color *k*: c(*j*)=*k*
10. $U = U - \{j\}$, $C = C \cup \{u\}$

ENDWHILE

Figure 2. Constructive method C3

In Section 5 we will compare the three constructive methods C1, C2 and C3 with each other and with the memory-based methods proposed in the following subsection.

3.2 Memory-Based Constructive Methods

In this subsection we consider a search framework given by the use of memory among constructions. Instead of performing an independent sampling of the solution space, constructive methods based on memory structures perform a guided selection in this space (a seminal reference can be found in Fleurent and Glover 1999). Specifically, in tabu search constructions, the inclusion or exclusion of certain elements or groups of elements can be identified as attractive for intensification or diversification purposes. In this section we focus our attention on constructive and destructive neighborhoods in which an element is added to or dropped from the partial solution under construction.

In this context, we consider the inclusion of frequency memory to modify the evaluations of a greedy function.

To record relevant information common to different constructions in the BCP, it must be noted that the color $c(i)$ assigned to a vertex i is not a relevant information by itself since different colorings can represent the same BCP solution. Consider, for example a bipartite graph with all the edge weights equal to 1 in which all the vertices in the first partition are colored with color 1 and all the vertices in the second partition are colored with color 2. Now consider another coloring of the same graph in which we switch the colors between partitions (i.e., all the vertices in the first partition are colored with color 2 and all the vertices in the second partition are colored with color 1). These two colorings do not have any vertices with the same color; however as solutions of the BCP they can be considered the same. We therefore will focus on the relative colorings between adjacent vertices, instead of the color of each single vertex.

We record in $freq[i][j]$ the number of times (i.e., the number of previous constructions) in which vertices i and j have been colored with similar colors (i.e., colors with close numbers). Bandwidth constraints force $c(i)$ and $c(j)$ to verify $|c(i)-c(j)| \geq d_{ij}$ and, in some cases, these constraints can be accomplished with equality: $|c(i)-c(j)| = d_{ij}$. The importance of these *binding constraints* is well document in the mathematical programming literature. We extend this situation to relatively close colors and say that i and j are *binding vertices* in a solution if the absolute difference of their colors is close to the associated edge weight. In mathematical terms:

$$If \;\; |c(i) - c(j)| - d_{ij} \leq bind\_th \quad then \;\; freq[i][j] = freq[i][j] + 1$$

where *bind_th* is a search parameter (a threshold value) to measure the distance between adjacent colors with respect to the edge weight. We will study different values for this parameter in our computational experiments.

1. Initially $C = \emptyset$, $U=V$ and $k=1$.
2. Select a vertex i randomly from U.
3. Assign the color 1 to i ($c(i)=1$). $C = \{i\}$, $U = U - \{i\}$

WHILE ($U \neq \emptyset$)

4. Compute $eval\_m(i)$ for all i in U
5. Select the vertex i^* with minimum $eval\_m$ value in U
6. Let h be the first (minimum) legal color for i^*
7. Label vertex i^* with color h: $c(i^*)=h$
8. If ($h = k+1$) do $k=k+1$.
9. $U = U - \{j\}$, $C = C \cup \{u\}$
10. Update $freq[i][j]$ for all i, j.

ENDWHILE

Figure 3. Constructive method M1

M1 is basically a memory based sequential construction in which we first choose a vertex (in a greedy randomized fashion) and then assign an appropriate color. We modify the evaluation of the attractiveness of each non-colored vertex in the current construction according to the following quantities to favor the selection of new structures, which were not generated in previous iterations:

$$\text{M1: } eval\_m(i) = \min \left\{ c \in \mathbb{Z} : \; |c - c(j)| \geq d_{ij} + \beta \cdot freq[i][j] \;\; \forall j \in N(i) \cap C \right\}$$

The parameter β controls the distance between colors that we impose on those vertices which frequently have received close colors. Figure 3 shows a pseudo-code of M1, which can be viewed as the memory-based versions of the C1 method described in the previous subsection.

In our second memory-based constructive method, M2, we add memory structures to the method C3 in which a color is completed before introducing a new one. Specifically, we modify the definition of candidate list *CL* to incorporate the frequency information recorded in previous iterations:

$$CL = U \cap \{i \in V : |k - c(j)| \geq d_{ij} + freq[i][j] \quad \forall j \in N(i) \cap C\} .$$

In this way we only can assign a color to those vertices in which this assignment does not lead to a solution structure already generated (according to the *freq*-information). The ranking of the candidate vertices in *CL* is made, as in C3, according to the number of uncoloured adjacent; however, instead of randomly select one vertex in a restricted candidate list, here we select the best one. Figure 4 shows a pseudo-code of this method.

1. Initially $C = \emptyset$, $U=V$ and $k=1$.
2. Select a vertex i randomly from U.
3. Assign the color 1 to i ($c(i)=1$). $C = \{i\}$, $U = U - \{i\}$

WHILE $(U \neq \emptyset)$

4. $CL = U \cap \{i \in V : |k - c(j)| \geq d_{ij} + freq[i][j] \quad \forall j \in N(i) \cap C\}$

IF ($CL = \emptyset$)

5. $k = k + 1$

ELSE

6. Compute $eval(i) = |N(i) \cap U|$ for all i in CL
7. Select the vertex j with the lowest *eval*-value in CL
9. Label vertex j with color k: $c(j)=k$
10. $U = U - \{j\}$, $C = C \cup \{u\}$

ENDWHILE

Figure 4. Constructive method M2

4. Improvement Methods

In this section we explore two different types of improvement methods: local search and short-term tabu search. The first ones are typically embedded in GRASP algorithms so they naturally can coupled the GRASP constructions proposed in Section 3.1. Symmetrically, the short term tabu search can be implemented as a post-processing to the memory-based constructive methods described in Section 3.2. Other types of combinations between constructions and improvements are also tested in our computational experiment described in Section 5.

4.1 Local Search Methods

A classical local search method in graph coloring problems consists of reducing by one unit the number of colors in a given solution, trying to obtain a better feasible solution. Our first local search method, LS1, uncolors all the vertices with maximum color, say *kmax*, and re-colors them with a color in [1, *kmax*-1]. The method selects the best available color for these vertices minimizing the violation of the bandwidth constraints. In mathematical terms, given a vertex i with $c(i)=kmax$, the method assigns $c(i)=c_{best}$ where c_{best} minimizes

$$F(c) = \sum_{j \in N(i)} max\{d_{ij} - |c - c(j)|, 0\}$$

Once all these vertices have been re-colored, LS1 performs further iterations selecting their adjacent vertices in which the associated edge weight is larger than the absolute difference of colors (those causing unfeasibility). The method then re-colors these vertices in a similar way as in the previous iteration (i.e., assigning them a color in [1, *kmax*-1] minimizing the expression *F*(*c*)). LS1 continues in this way until all the vertices have a proper color (i.e., we obtain a feasible solution) or it reaches a maximum number of iterations. In the former case we reduce the number of colors by one unit again and perform a new global iteration; otherwise the method stops.

Glover and Laguna (1997) introduced the *compound moves*, often called *variable depth methods*, constructed from a series of simpler components. One of the well-known pioneering contributions to such moves was the work by Lin and Kernighan (1973). Within the class of variable depth procedures, a special subclass called *ejection chain procedures* has recently proved useful. An ejection chain is an embedded neighborhood construction that compounds the neighborhoods of simple moves to create more complex and powerful moves. It is initiated by selecting a set of elements to undergo a change of state (e.g. to occupy new positions or receive new values). The result of this change leads to identifying a collection of other sets, with the property that the elements of at least one must be "ejected from" their current states. State-change steps and ejection steps typically alternate, and the options for each depend on the cumulative effect of previous steps (usually, but not necessarily, being influenced by the immediately preceding step). In some cases, a cascading sequence of operations may be triggered, representing a domino effect.

Our second local search method for the BCP, LS2, implements an ejection chain to reduce the number of colors in a solution by one unit. Let *kmax* be the number of the maximum color in a solution. The method first identifies the set of nodes, *C*(*kmax*, 0) colored with this color,

$$C(kmax, 0) = \{i \in V : c(i) = kmax\}.$$

Then it computes the set of vertices, *C*(*kmax*, 1), with binding constraints with the vertices in *C*(*kmax*, 0). In other words, those vertices that "force" the vertices in *C*(*kmax*, 0) to take the *kmax* color:

$$C(kmax, 1) = \{i \in V : |c(i) - c(j)| = d_{ij} \ , \ j \in C(kmax, 0)\}.$$

The rationale behind this is that we should modify the color of the vertices in *C*(*kmax*,1) to reduce the color of the vertices in *C*(*kmax*, 0). We could say that they are one step apart. We proceed likewise to compute the vertices *C*(*kmax*, 2) that force the vertices in *C*(*kmax*, 1) to have their color:

$$C(kmax, 2) = \{i \in V : |c(i) - c(j)| = d_{ij} \ , \ j \in C(kmax, 1)\}.$$

Vertices in *C*(*kmax*, 2) are two steps apart from vertices in *C*(*kmax*, 0). Likewise we compute *C*(*kmax*, *k*) for *k*=0, 1, 2,.., *depth*, where *depth* is a search parameter of the ejection chain. We then proceed to modify the color of the vertices in these sets from

k=0 to *depth*. We assign a new color h to vertex $i \in C(kmax, k)$ considering that the vertices in $C(kmax, k+1)$ have a dummy color (with value 0) and computing

$$h = \min \{c \in \mathbb{Z} : |c - c(j)| \geq d_{ij} \ \forall j \in N(i)\},$$

which enables us to reduce the color of the vertices in $C(kmax, k)$. In the final step the method colors the vertices in $C(kmax, depth)$. If it succeeds, we finally obtain a feasible solution with *kmax*-1 colors and perform a new complete iteration to further reduce the number of colors. In some cases we do not need to examine the entire C sets (from 0 to *depth*) because after examining some of them we obtain a feasible solution. In this event we stop at that point and resort to a new global iteration. Similarly, if a vertex $i \in C(kmax, k)$ re-colored during the ejection chain, also belongs to another C set ($i \in C(kmax, k+p)$ for some p) we stop the ejection chain process to avoid cycling. It is worth mentioning that if we set a conservative value of *depth* (close to 10) the method always stops because of these last two situations and therefore we do not need to adjust this parameter. Thus the ejection chain works in practice as a reactive mechanism terminated by its internal logic.

4.2 Tabu Search Methods

Malaguti and Toth (2007) recently proposed a short term tabu search, T1, for the BCP based on partial solutions (in which not all the vertices are colored). In any global iteration, T1 tries to obtain a solution with *kmax* colors. Given a constructed solution, the method first un-colors the vertices with a color $k > kmax$; then it performs successive steps to color these vertices. At any local iteration the method randomly selects an uncolored vertex i and colors it with a color $h \in [1, kmax]$. Then, it un-colors all its adjacent vertices j violating the bandwidth constraint (i.e., $c(j)$=0 for all j in $N(i)$: $|c(i)-c(j)| \leq d_{ij}$). Color h is selected to minimize the sum of edge weights incident with uncolored vertices. The tabu status indicates that uncolored vertices cannot take the same color they had for a certain number of consecutive iterations. When all the vertices are colored, T1 reduces *kmax* by one unit and performs a new global iteration.

Now we will propose a tabu search improvement method, T2, based on LS2, the ejection chain local search procedure. Specifically, we compute $C(kmax, k)$ for k=0, 2,.., *depth*, (where *depth* is the parameter of the ejection chain) but introducing a short-term memory design in which the identity of a vertex whose color has been changed is the attribute used to impose a tabu restriction. Specifically, after a move is executed, the colors of the vertices involved in the move are not allowed to change until the tabu tenure expires. We employ a one-dimensional array *tabu*(i) initially set to zero, to store the iteration number when vertex i loses its tabu status. That is, if vertex i changes colors at iteration *iter*, then *tabu*(i)=*iter*+*tenure*, where *tenure* is the number of iterations that vertex i is not allowed to change colors.

Note that although we can use the same *tenure* value for all vertices involved in the move, an interesting variant is to use a different *tenure* value for different types of vertices. In such a design, the *tenure* value for vertex $i \in C(kmax, k)$ could be different from the *tenure* value for vertex $j \in C(kmax, k+1)$, because they have different roles in the computation of the *kmax* value of the current solution. We have not implemented this variant since preliminary experiments indicate that it significantly increases the complexity in order to calibrate the additional search parameter.

5. Computational Experiments

This section describes the computational experiments that we have performed to first test the efficiency of our different procedures and then compare them with a number of methods from the literature. We have implemented the methods in Java SE 6 and all the experiments were conducted on a Pentium 4 computer at 3 GHz with 2 GB of RAM.

We have employed the set GEOM with the instances reported in most of the previous BCP papers. GEOM consists of 33 geometric graphs generated by Michael Trick (and available at http://mat.gsia.cmu.edu/COLOR02/). In these graphs, points are generated in a 10,000 by 10,000 grid and are connected by an edge if they are close enough together. Edge weights are inversely proportional to the distance between nodes. This set contains three types of graphs. The GEOMn instances are sparse, the GEOMa and GEOMb instances are denser, where GEOMb requires fewer colors per node.

In each experiment we compute for each instance and each method the relative percent deviation (*Dev.*) between the best solution value (*Value*) obtained with the method and the best known value (*BestValue*) for that instance. Best known values were obtained in Malaguti and Toth (2007) with extremely long running times of their evolutionary method (with CPU times not reported in the paper but estimated in several hours). We report the average of *Dev.* and *Value* across the 33 instances considered in each particular experiment. For each method, we also report the number of instances (*#Best*) for which the value of the solution obtained with this method is the best one in this particular experiment (although it does not necessarily match *BestValue*) and the average CPU in seconds (*Time*) that it consumes in each experiment.

bind_th	**β**	***Dev.***	***#Best***	***Time***
3	0.1	31.25	5	0.0012
3	0.2	33.14	4	0.0013
3	0.3	33.20	0	0.0015
4	0.1	31.19	3	0.0012
4	0.2	32.93	2	0.0014
4	0.3	34.06	2	0.0015
5	0.1	31.06	3	0.0017
5	0.2	32.05	3	0.0014
5	0.3	33.15	2	0.0015

Table 1. Preliminary experimentation for M1.

The preliminary experimentation was performed on 10 representative problem instances with the goal of finding appropriate values for the key search parameters of M1 and C3. We tested values for *bind_th* in the range [3, 5], β in [0.1, 0.3] and α in [0, 1]. The results of running method M1 for 100 constructions are shown in Table 1. They indicate that *bind_th*=5 and β=0.1 provide the best results for these methods. Hence, we use these values to perform the rest of our experimentation.

α	*Dev.*	*#Best*	*Time*
0.0	30.98	4	0.004
0.1	32.17	1	0.004
0.2	31.40	1	0.004
0.3	31.55	4	0.004
0.4	31.69	3	0.004
0.5	31.57	5	0.004
0.6	30.22	5	0.004
0.7	31.05	3	0.004
0.8	31.48	2	0.004
0.9	31.31	1	0.004
1.0	32.03	3	0.004

Table 2. Preliminary experimentation for C3.

Our second preliminary experiment tests different values for α in the GRASP construction C3. Table 2 shows that there are no significant differences among the results obtained with the different α values. Therefore, in the GRASP constructions C1, C2 and C3 we will employ the strategy recommended in Resende et al. (2009) consisting of randomly selecting an α value in each construction.

In our first final experiment we compare the two previous constructive methods, SEQ (Lim et al. 2005) and DSATUR (Malaguti and Toth 2008), with our three memory-less variants, C1, C2 and C3, and with the two memory-based methods, M1 and M2. We run each method to generate 100 solutions for each of the 33 problems. Table 3 reports the statistics *Dev.* and *#Best* described above for these seven methods. In addition, we calculate the *Rank* statistic — proposed by Ribeiro, et al. (2002) — associated with each method. For each instance, the *n_rank* of a method M is defined as the number of methods that found a better solution than the one found by M. In the event of ties, all the methods receive the same *n_rank*, equal to the number of methods strictly better than all of them. The value of *Rank* is the sum of the *n_rank* values for all the instances in the experiment, thus, the lower the *Rank* the better the method.

	Method	***Dev.***	***#Best***	***Rank***	***Time***
Previous	SEQ	56.55	7	205	0.000
	DSATUR	47.75	2	174	0.077
Memory-less	C1	25.65	27	56	0.030
	C2	49.18	1	182	0.075
	C3	25.69	25	55	0.002
Memory-based	M1	26.25	22	64	0.046
	M2	35.69	1	127	0.002

Table 3. Constructive methods

Results in Table 3 clearly show that the new constructive methods outperform the previous ones. In addition, all of them are extremely fast, and exhibit running times of much less than 1 second. Moreover, methods C1, C3 and M1 give the best results from among the seven methods tested.

	Method	***Dev.***	***#Best***	***Rank***	***Time***
Memory-less	LS1	13.92	3	60	0.3
	LS2	14.51	1	71	0.9
Memory-based	T1	13.86	2	61	14
	T2	13.83	3	63	25

Table 4. Improvement methods

Our second experiment is designed to isolate and compare the contribution of the improvement methods. We therefore generate the initial solutions using a relatively simple constructive method, SEQ, and apply the different improvement methods to them. Table 4 reports the results obtained with the four improvement methods described in Section 4: LS1, LS2, T1 and T2 when they were run for 100 constructions + improvements. After preliminary adjustment we set *tenure=4* and *maxIterations*=50.

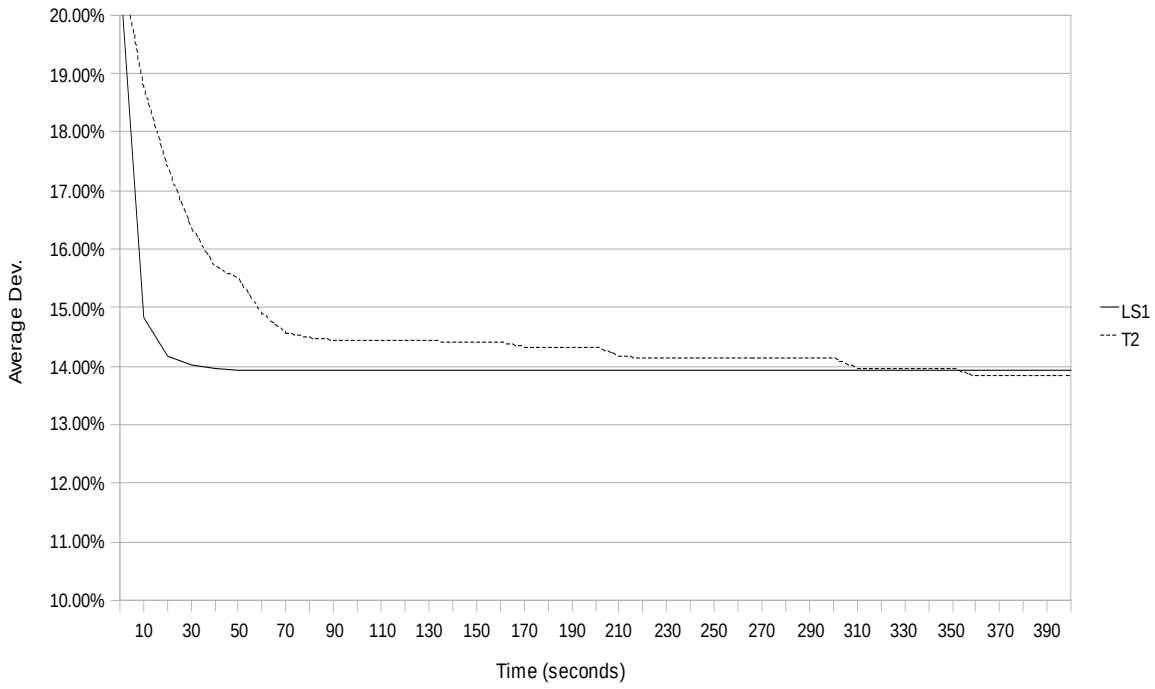

Figure 5. Search profile

Table 4 shows that memory-based methods perform slightly better than memory-less methods, although they consume more CPU time. Comparing the results in this table with the results of the constructive methods in Table 3 we observe that the improvement methods reduce the average deviation by about 15% on average. We complement the results shown in Table 4 with a comparison of the performance of the best improvement methods, LS1 and T2, over time. These two methods were run for 500 seconds and the best solution found was reported every 10 seconds. The results of this experiment are

shown in Figure 5. It shows that LS1 is capable of obtaining good quality solutions from the very beginning of the search (i.e., within the first 100 seconds). T2 requires 350 seconds to improve upon the solutions found by LS1 but it then maintains its lead during the remaining execution time (although we do not depict in the diagram the last 100 seconds for the sake of clarity, both methods present flat profiles in that part).

We have also considered the evolution of LS2 and T2 in order to see the effect of the memory structure. Figure 6 depicts the search profile of both methods. This figure clearly shows that the addition of memory structures permits better solutions to be obtained (compared with the memory-less variant of the same method) from the very beginning of the search and, what is more important, the memory-less variant is unable to improve the other method at any point.

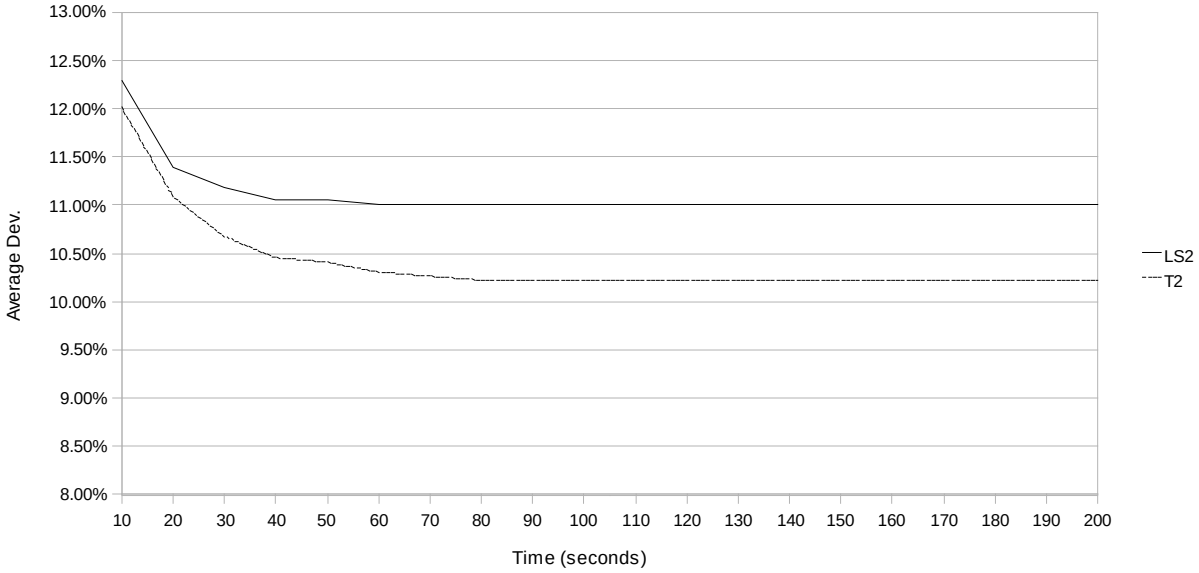

Figure 6. Search profile

In the next experiment we combine the best constructive methods with the best improvement methods. Specifically we combine C1, C3 and M1 with LS1, LS2 and T2 and run the 9 resulting methods for 1000 constructions + improvements. Table 5 shows the results of this experiment.

	Method	***Dev.***	***#Best***	***Rank***	***Time***
C1	LS1	5.55	10	106	6
	LS2	4.95	16	56	3
	T2	4.89	17	53	3
C3	LS1	6.27	12	149	3
	LS2	4.08	29	25	2
	T2	4.03	30	24	3
M1	LS1	6.23	12	161	31
	LS2	5.04	17	65	6
	T2	5.25	16	92	6

Table 5. Construction + Improvement methods

Table 5 shows that C3+T2 is the best combination of our constructive with improvement methods since it obtains 30 best solutions, out of 33 instances, and 4.03% deviation with respect to the best known solution. All of them are very fast since they construct and improve 1000 solutions in a few seconds, with the exception of M1+LS1, which consumes about half a minute on average.

We applied the Friedman test for paired samples to the best solutions obtained by each method. This test computes, for each instance, the rank-value of each method according to solution quality (where rank 1 is assigned to the best method and rank 9 to the worst one). Then, it calculates the average rank values of each method across all the instances solved. If the averages differ greatly, the associated *p*-value or significance will be small. The resulting significance level of 0.000 obtained in this experiment clearly indicates that there are statistically significant differences among the nine methods tested. Specifically, the rank values produced by this test are C3+T2 (3.18), C3+LS2 (3.29), C1+T2 (4.71), M1+LS2 (4.76), C1+LS2 (4.83), M1+T2 (5.35), C1+LS1 (5.86), C3+LS1 (6.38) and M1+LS1 (6.64). This confirms that among the procedures that we tested, C3+T2 is the best to obtain the highest quality solutions.

In the final experiment we compare our best method with three previous methods identified as the best. Specifically, we consider the following four methods:

- AMP : our best method, C3+T2
- DSATUR+T1: the constructive + improvement by Malaguti and Toth (2008)
- FCNS: the constraint propagation method by Prestwich (2002)
- Multi-start: the constructive + improvement by Lim et al. (2005)

We can see the results of this final experiment in Table 6. It is clear that our methods are competitive with the state of the art methods for this problem. We also applied a statistical test to the data used to generate Table 6.

Method	***Dev.***	***#Best***	***Rank***	***Time***
AMP	3.38	13	13	11.00
DSATUR +T1	5.88	9	50	18.42
FCNS	2.62	12	7	12.91
Multi-start	5.04	13	34	3.83

Table 6. Best Constructions + Improvement methods

Considering the deviations from the best solution known, the FCNS appears to the best one. On the other hand, considering the number of best solutions, our AMP is ranked first together with the Multi-start method. The Friedman test obtains a significance level of 0.000 indicating that there are statistically significant differences among the four methods tested. Specifically, the rank values produced by this test are FCNS (1.97), AMP (2.17), Multi-start (2.76) and DSATUR+T1 (3.11). We can conclude that the four methods considered are able to obtain high quality solutions for the bandwidth coloring problem, where the FCNS appears to be the best, closely followed by our AMP method.

6. Conclusions

The objective of our study has been to compare memory-based with memory-less designs. We have developed heuristic procedures based on both the GRASP and the tabu search methodologies to provide high quality solutions to the bandwidth coloring problem. Unlike local search methodologies, memory structures have not been extensively studied yet in the context of constructive methods. In this paper we have proposed two constructive methods based on incorporating basic tabu search memory structures, and three memory-less constructions based on GRASP methodology. These five procedures have been coupled with an improving phase. We have also tested the inclusion of memory (short-term structures) in the improvement phase. The final procedures, labeled as Adaptive Memory Programming methods, are able to compete with the state-of-the-art methods for the bandwidth coloring problem within short computational times.

Our findings disclose the novel fact that memory appears to play a more important role during the improvement phase than during the constructive phase of search. This effect may be due to the fact that the repeated application of the constructive phase operates primarily as a diversification process, and that more advanced memory-based mechanisms, including statistical analysis as suggested in Glover and Laguna (1997), may be needed to provide advantages for a combined construction/improvement procedure. This issue provides an interesting area for future research.

Acknowledgments

The authors want to thank Prof. Fred Glover for his help with the AMP descriptions. This research has been partially supported by the Ministerio de Educación y Ciencia of Spain (TIN2006-02696) and by the Comunidad de Madrid—Universidad Rey Juan Carlos project (CCG08-URJC/TIC-3731).

References

Brélaz, D. (1979) “New Methods to Color Vertices of a Graph”, *Communications of the ACM* 22(4), 251-256.

Duarte, A. and R. Martí (2007) “Tabu Search for the Maximum Diversity Problem”, *European Journal of Operational Research* 178, 71-84.

Fleurent, C. And F. Glover (1999) “Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory”, *INFORMS Journal on Computing* 11(2), 198-204.

Garey, M.R. and D.S. Johnson (1979) *Computers and Intractability. A guide to the theory of completeness*, W. H. Freeman and Company, New York.

Glover, F. (1996) Tabu Search and Adaptive Memory Programming - Advances, Applications and Challenges, in: *Interfaces in Computer Science and Operations Research*, Barr, Helgason and Kennington (eds.) Kluwer Academic Publishers, 1-75.

Glover, F. and M. Laguna (1997) *Tabu Search*, Kluwer Academic Publishers, Boston.

Laguna, M. and R. Martí (1999) “GRASP and Path Relinking for 2-Layer Straight Line Crossing Minimization”, *INFORMS Journal on Computing*, vol. 11, no. 1, pp. 44-52.

Laguna, M. and R. Martí (2003) "*Scatter Search: Methodology and Implementations in C*", Kluwer Academic Publishers, Boston.

Lim, A., Y. Zhu, Q. Lou and B. Rodrigues (2005) "Heuristic Methods for Graph Coloring Problems," *The 20th Annual ACM Symposium on Applied Computing* (SAC 2005, Santa Fe, New Mexico) March 13-17.

Lin, S. and B. Kernighan (1973), An effective heuristic algorithm for the traveling salesman problem, *Operations Research* 21, 498-516.

Malaguti, E. and P. Toth (2008) "An evolutionary approach for bandwidth multicoloring problems", *European Journal of Operational Research 189, 638-651*.

Martí, R., M. Laguna, F. Glover and V. Campos (2001) "Reducing the Bandwidth of a Sparse Matrix with Tabu Search", *European Journal of Operational Research*, 135 (2), 211-220.

Phan, V. and S. Skiena (2002) "Coloring graphs with a general heuristic search engine" Computational symposium on graph coloring and its generalizations, Ithaca, NY, 92-99.

Prestwich, S. (2002) "Constrained bandwidth multicoloration neighborhoods" Computational symposium on graph coloring and its generalizations, Ithaca, NY, 126-133.

Resende, M., R. Martí, M. Gallego and A. Duarte (2009) "GRASP and Path Relinking for the Max-Min Diversity Problem", to appear in *Computers and Operations Research*.

Resende, M.G.C. and C.C. Ribeiro (2003) "Greedy Randomized Adaptive Search Procedures", in *Handbook of Metaheuristic*, Kluwer Academic Publishers, 219–249.

Ribeiro, C. C., E. Uchoa and R. F. Werneck (2002) "A Hybrid GRASP with Perturbations for the Steiner Problem in Graphs," *INFORMS Journal on Computing* 14, 228 – 246.

Welsh, D. J. A. and M. B. Powell (1967) "An Upper Bound for the Chromatic Number of a Graph and its Application to Timetabling Problems," *Comput. J.*, 10, 85-86.