Carousel Greedy: A Generalized Greedy Algorithm with Applications in Optimization

Article *in* Computers & Operations Research · April 2017

DOI: 10.1016/j.cor.2017.03.016

CITATIONS
40

READS
4,305

3 authors:

Carmine Cerrone
Università degli Studi di Genova
34 PUBLICATIONS **291** CITATIONS
SEE PROFILE

Raffaele Cerulli
Università degli Studi di Salerno
104 PUBLICATIONS **1,156** CITATIONS
SEE PROFILE

Bruce Golden
University of Maryland, College Park
366 PUBLICATIONS **18,260** CITATIONS
SEE PROFILE

Carousel Greedy: A Generalized Greedy Algorithm with Applications in Optimization

Carmine Cerrone
Department of Biosciences and Territory
University of Molise
carmine.cerrone@unimol.it

Raffaele Cerulli
Department of Mathematics
University of Salerno
raffaele@unisa.it

Bruce Golden
Robert H. Smith School of Business
University of Maryland
bgolden@rhsmith.umd.edu

Abstract

In this paper, we introduce *carousel greedy*, an enhanced greedy algorithm which seeks to overcome the traditional weaknesses of greedy approaches. We have applied carousel greedy to a variety of well-known problems in combinatorial optimization such as the minimum label spanning tree problem, the minimum vertex cover problem, the maximum independent set problem, and the minimum weight vertex cover problem. In all cases, the results are very promising. Since carousel greedy is very fast, it can be used to solve very large problems. In addition, it can be combined with other approaches to create a powerful, new metaheuristic. Our goal in this paper is to motivate and explain the new approach and present extensive computational results.

1 Introduction

Greedy algorithms have been developed for a large number of problems in combinatorial optimization. For many of these greedy algorithms, elegant worst-case analysis results have been obtained. These greedy algorithms are typically very easy to describe and code and they have very fast running times. On the other hand, the accuracy of greedy algorithms is often unacceptable.

When instance size is large, exact solution approaches are generally not practical. However, a wide variety of metaheuristics (e.g., tabu search, genetic algorithms, variable neighborhood search, etc.) have been successfully applied to solve combinatorial optimization problems to within 5% (or better) of optimality. In contrast to greedy algorithms, metaheuristics can be complex and difficult to code. Running times can grow quickly with instance size. One key question that arises is as follows: Is there a middle ground between greedy algorithms and metaheuristics? In other words, can we identify a class of heuristics that performs nearly as well as metaheuristics with respect to accuracy, but has

running times (and the simplicity) that are similar to what we see with greedy algorithms?

In this paper, we seek to answer these questions. In particular, we propose the *carousel greedy algorithm*. The carousel greedy algorithm is an enhanced greedy algorithm which, in comparison to a greedy algorithm, examines a more expansive space of possible solutions with a small and predictable increase in computational effort. We demonstrate the utility of the carousel greedy approach by applying the procedure to several well-known optimization problems defined on graphs, as well as an important problem in statistics. An outline of the carousel greedy algorithm (CG) is as follows:

(i) Create a partial solution using a greedy algorithm;

(ii) Use the same greedy algorithm to modify the partial solution in a deterministic way;

(iii) Apply the greedy algorithm to produce a complete, feasible solution.

Unlike a typical metaheuristic, CG does not progress from one feasible solution to another, hopefully better, feasible solution. Only one feasible solution, at the end of the algorithm, is identified. In fact, CG does not calculate an objective function until the final step. One can consider the greedy and the CG algorithms to be constructive metaheuristics in contrast to improving metaheuristics.

We will argue that carousel greedy can be applied to many types of decision problems. In order to apply CG to a specific decision problem, we assume that there already exists a constructive greedy algorithm for that problem; otherwise, CG may be more difficult to apply. Our primary focus in this paper will be on problems involving the minimization or maximization of the cardinality of a set. In particular, we will study the following four problems: The minimum label spanning tree, the minimum vertex cover, the maximum independent set, and the minimum weight vertex cover. Although it is not difficult to imagine how carousel greedy can be applied more generally, we believe that these four problems are natural beneficiaries of carousel greedy; our focus on them will allow us to easily illustrate the characteristics and potential of the carousel greedy algorithm.

Our paper is organized as follows. In Section 2, we provide motivation. The carousel greedy algorithm is presented in detail in Section 3. In Section 4, computational experiments in combinatorial optimization are described. In Section 5, a Hybrid carousel greedy algorithm is presented. In Section 6, conclusions and future work are discussed.

2 Motivation

As mentioned in Section 1, we seek a heuristic framework that generalizes and enhances greedy algorithms. We want a heuristic that is fast. It should routinely outperform a greedy algorithm with respect to accuracy. It should be applicable to many greedy algorithms. In addition, it should be simpler than a metaheuristic and it should involve a small number of parameters.

At this point, we introduce the minimum label spanning tree ($MLST$) problem. We will use this specific problem to help motivate the carousel greedy algorithm. In the $MLST$ problem, we are given an undirected graph with labeled edges as input. We can think of each label as a color or a letter. Each edge has a label and different edges can have the same label. We seek a spanning tree with the minimum number of distinct labels (see Appendix G).

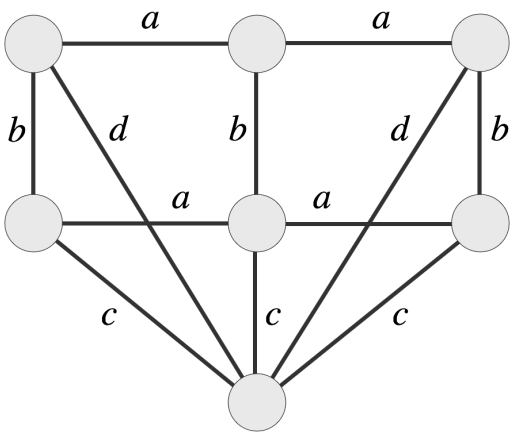

Figure 1: A small $MLST$ instance

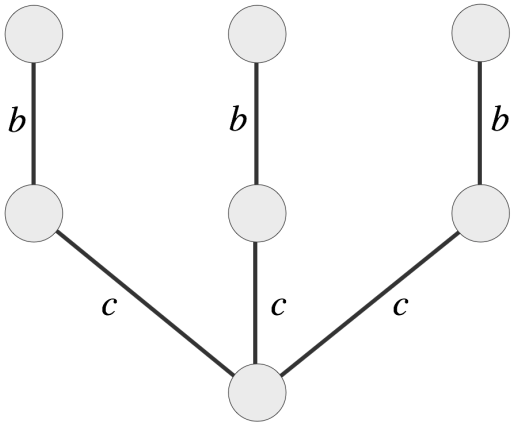

Figure 2: The optimal solution

Chang and Leu [8] introduced the problem which has applications to communications network design. They proved the $MLST$ problem is NP-hard and proposed a greedy heuristic called the maximum vertex covering algorithm ($MVCA$). Their heuristic, however, contained a major error. A corrected version of $MVCA$ was proposed by Krumke and Wirth [29]. This correct version of $MVCA$ begins with a graph consisting of nodes only. It adds labels (and, therefore, edges) by reducing the number of connected components by as much as possible, until only one connected component remains. Of course, any spanning tree constructed on this connected component will be a solution to the problem. A detailed description of $MVCA$ is provided in Appendix A. A variety of metaheuristics for the $MLST$ problem have been presented by Cerulli et al. [7], Xiong et al. [35, 36], Consoli et al. [11, 12, 13], and others.

Now let's consider the small $MLST$ instance in Figure 1. The letters represent the labels. If we apply $MVCA$, we add labels a, b, and then c to obtain $\{a, b, c\}$ and the corresponding edges as a solution. The optimal solution is shown in Figure 2.

The key observation here is that $MVCA$ chose a wrong label (namely, a) initially and had no way to overcome or correct this mistake. The carousel greedy algorithm is designed with this failure in mind.

Based on our experience with $MVCA$ (and greedy algorithms in general), we divide $MVCA$ into three phases (see Figure 3). In Figure 3, $|S|$ represents the number of labels selected by $MVCA$. In Phase I, the set of candidate labels is very large. Many labels yield similar results. So, it is difficult to know which label to select. In addition, it is not possible to learn much from previous label selections, because there have not been many.

In Phase II, the set of candidate labels has been reduced. It is possible to learn from previous label selections. This is because each remaining candidate label, if selected, would have an incremental contribution (in terms of the number of connected components) which is dependent on previous label selections. In this sense, the Phase II selections are *smarter* than the Phase I selections. Some labels, chosen in Phase I, no longer look so good (e.g., see Figures 1 and 2). These observations are consistent with the Principle of Marginal Conditional Validity (see Glover [19]):

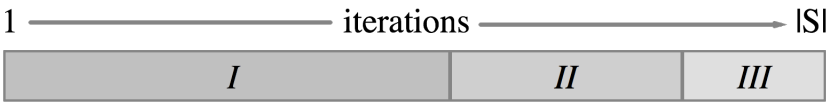

Figure 3: Dividing $MVCA$ into three phases

> "As more decisions are made in a constructive approach (as by assigning values to an increasing number of variables), the information that allows these decisions to be evaluated becomes increasingly valid, conditional upon the decisions previously made."

An immediate implication of this principle is that early decisions are likely to be inferior to later decisions. That is, in some cases, myopic early decisions may constrain later choices to be less than satisfactory.

By the end of Phase II, nearly all the labels that $MVCA$ will select have been selected. In Phase III, the set of candidate labels required for feasibility is small. Therefore, Phase III is short.

2.1 Experimental Justification

In order to test our intuition on the three phases of a greedy algorithm, we designed a small experiment. The experiment involves the $MLST$ problem, described in Section 2. In particular, we generated $10,000$ random labeled graphs with 81 nodes, 160 edges, and 40 labels.

These graphs are generated as described by Xiong et al. [35]; they are referred to as Group II graphs and they are constructed in such a way that the optimal solution is known in advance. The optimal solution is 20 labels for each of $10,000$ graphs in this experiment. Using a mathematical programming model derived from Captivo et al. [4], we verified that the optimal solution is unique for each instance. We generate each of the test graphs as follows:

(i) Generate a graph according to [35] with 81 nodes, 40 labels, and 4 edges for each label;

(ii) This specifies an optimal solution of size 20;

(iii) Starting with the formulation from [4], add a constraint that limits the sum of all binary variables associated with the 20 labels in the optimal solution to be less than 20;

(iv) If the model produces a new solution with 20 labels, then the optimal solution is not unique and we reject the graph;

(v) Otherwise, the optimal solution is unique and we use the graph.

For each graph, $MVCA$ is applied. Averaging over the $10,000$ applications of $MVCA$, we obtain the results presented in Figure 4. The *x-axis* indicates the percentage of the selections in $MVCA$ that have been completed.The *y-axis* indicates the average percentage of selected labels up to the current point in $MVCA$ that turn out to be in the optimal solution.

Upon close inspection of Figure 4, we observe the following:

a) In Phase I, the performance is steady. The average percentage of labels in the optimal solution is less the 55%.

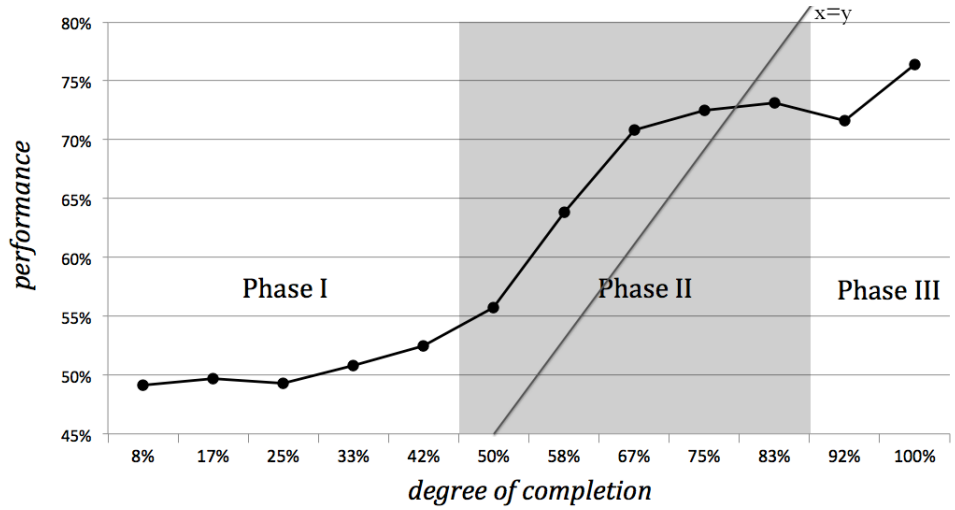

Figure 4: $MVCA$ phase experiment. The graph indicates the average percentage of selected labels that turn out to be in the optimal solution, at any degree of completion of $MVCA$.

b) In Phase II, the performance is increasing. The average percentage of labels in the optimal solution is above 55% and increasing.

c) In Phase III, the performance increases rapidly, after an initial decline. The average percentage of labels in the optimal solution increases to above 77%.

In a second experiment, we wanted to study the impact of the cardinality of the candidate set of labels on the quality of the $MVCA$ solution. We expect $MVCA$ to make some poor selections in Phase I. One might conjecture that the larger the candidate set, the less impact these early mistakes will have on the overall performance of $MVCA$. The purpose of this experiment is to test such a conjecture.

We generated $1,300$ random label graphs with 81 nodes organized in 13 different groups, each one with 100 graphs. The number of labels, $|L|$, ranged from 30 to 390 in increments of 30. The number of edges was set equal to $4|L|$. Again, the graphs were Group II graphs from Xiong et al. [35]. The optimal solution is 20 labels in all cases.

The results of the second experiment are presented in Figure 5. The *x-axis* indicates the number of available labels. The *y-axis* indicates the average number of labels in the $MVCA$ solution. The average is taken over 100 graphs. It seems clear from Figure 5 that when $|L|$ is small, any mistake really hurts $MVCA$. When $|L|$ is large, maybe there are enough other labels to compensate for early errors. In any case, the second experiment reveals a major limitation of $MVCA$ and begs the question: Can a generalized greedy algorithm do better?

Based on the two experiments, several specific questions regarding the design of a generalized greedy algorithm come to mind:

1. Can we do more of Phase II and less of Phase I ?

2. Can we improve performance when $|L|$ is small ?

3. Can we ensure reasonable running times?

4. Can we keep it simple?

3 The Carousel Greedy Algorithm

Before we introduce the carousel greedy algorithm, it is important to acknowledge that the notion of a generalized greedy algorithm is not new. Duin and Voss [15] introduced the Pilot method in 1999 and applied it to the Steiner problem in graphs. Cerulli et al. [7] applied the Pilot method to the $MLST$ problem in 2005. Xiong et al. [36] proposed two modified versions of the Pilot method and applied these to the $MLST$ problem in 2006. The basic idea behind the Pilot method is to apply a greedy algorithm repetitively, each time from a different starting point (e.g., a different label).

The notion of a randomized greedy algorithm has been around for decades. Xiong et al. [36] applied a randomized $MVCA$ ($RMVCA$) to the $MLST$ problem in 2006. Instead of selecting the most promising label at each step, as in $MVCA$, consider the three most promising labels as candidates. Select one of the three labels probabilistically. We can run $RMVCA$ multiple times for each instance and output the best solution. This idea is, of course, related to $GRASP$, a well-known metaheuristic dating back to 1989 (see Feo and Resende [16, 17]).

Ruiz and Stützle [31] introduced the iterated greedy algorithm in 2007 and applied it to the permutation flowshop scheduling problem. The idea is both simple and general. The basic steps are provided below:

Step 1. Apply greedy algorithm to obtain a feasible solution.

Step 2. *Destruction phase*: Remove some elements from the current solution, leaving a partial solution.

Step 3. *Construction phase*: Apply the greedy algorithm to the partial solution to obtain another feasible solution.

Step 4. Repeat Steps 2 & 3 until a stopping condition is satisfied.

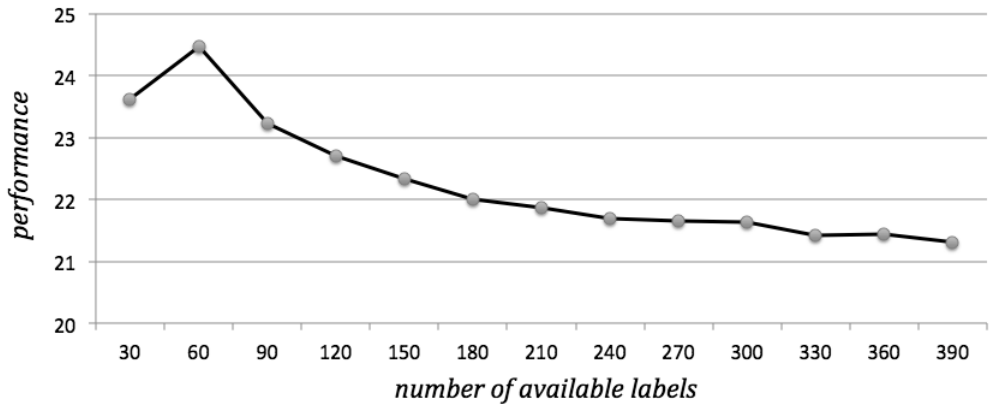

Figure 5: A limitation of $MVCA$. The graph indicates the average number of labels in the MVCA solution, with respect to the number of available labels (20 is the optimal solution for all the instances).

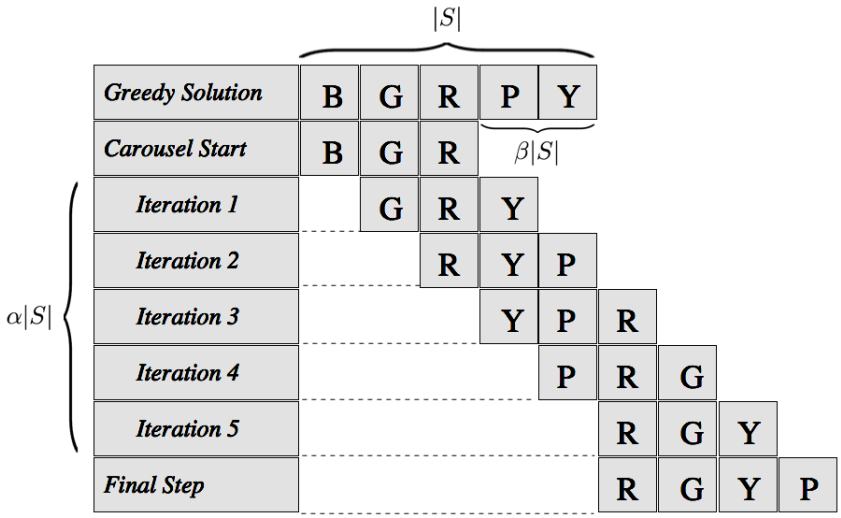

Figure 6: Carousel greedy illustration for $|S| = 5$, $\alpha = 1$, and $\beta = 40\%$

The Pilot method, randomized greedy, and iterated greedy are all useful and easy to implement. Each of the three generates a number of feasible solutions. The carousel greedy algorithm is also useful and easy to implement, however, it generates a single feasible solution. Because of this, we should expect carousel greedy to be much faster than other generalized greedy algorithms. This will enable us to handle huge problem instances. Moreover, it is designed specifically to address the four questions posed at the end of Section 2.1.

We will now introduce the carousel greedy algorithm using the schematic displayed in Figure 6. CG requires that two parameters, α and β, be specified, where α is an integer and β is a percentage. We will discuss these later. We can think of the letters as labels or colors ($B = black$, $G = green$, $R = red$, $P = purple$, and $Y = yellow$). Figure 6 illustrates how carousel greedy (CG) works. Each row is a step in the CG algorithm. We will now explain the figure, row by row.

In row 1, we begin with an execution of the greedy algorithm ($MVCA$ in this case). $MVCA$ selects $|S| = 5$ labels; these are B, G, R, P, and Y, in sequence of selection. In row 2, we drop the last $\beta = 40\%$ of the labels selected in the first row to leave us with B, G, and R. The idea is to postpone the last 40% or so of selections up to the very end. Until then, we will maintain a set of $(1 - \beta)|S|$ labels from row to row. Essentially, we are prolonging Phase II. Before moving on to row 3, we drop the first label in row 2 (namely, B). Next, given the remaining labels, G and R, we apply $MVCA$ to select a third label, Y, yielding G, R, and Y in row 3. Now, we drop G from row 3, apply $MVCA$ to select P, and obtain R, Y, and P for row 4. We continue in this manner until we reach row 7. At this point, we have labels R, G, and Y and we have completed 5 iterations since the start of carousel greedy. With $\alpha = 1$, in this example, we stop what is essentially an extended Phase II (note that α controls the degree to which Phase II is extended) after $\alpha|S| = 5$ iterations and proceed to what is essentially a short Phase III. In row 8, we start with the partial solution from row 7, apply $MVCA$ until we obtain a feasible solution, and then stop.

It is important to emphasize several key points. First, CG generates only one feasible solution, at the very end (we don't save the greedy solution). Second, it should be clear that early errors can

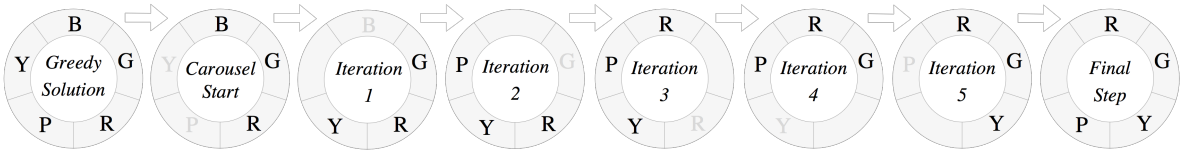

Figure 7: A second illustration of CG

be easily corrected, because we replace the labels selected in the early iterations using the greedy algorithm. Third, CG is only slightly more complicated than the original greedy algorithm. In this example, CG repeatedly calls upon $MVCA$. Also, in this case, the number of labels is reduced from 5 to 4.

There is another way to visualize and illustrate the carousel greedy algorithm. This second representation is the one that gives CG its name and it is presented in Figure 7. There are eight pictures in Figure 7 and the arrows indicate the sequence from start to finish. Figure 6 and 7 are complementary. In particular, the $i^{\underline{th}}$ picture in Figure 7 conveys the same information as does row i in Figure 6.

We can think of each picture in Figure 7 as a carousel which is divided into $|S|$ wedges. The first, second, and last of these carousels are stationary. The rest are in clockwise motion as the carousel makes α full turns or passes from iteration 1 to iteration 5 (we assume that α is an integer). A pseudo-code description of CG is provided in Appendix B.

Before discussing computational experiments, we estimate the computational complexity of CG with respect to the original greedy algorithm (denoted by G). Let $O(I)$ be the computational complexity of G. Also, let's assume that each iteration of G takes roughly the same amount of computational effort. Then, we can say that $M = \frac{O(I)}{|S|}$ represents the computational cost of a single iteration of G. Assume the cost of removing an element (e.g., a label) from the beginning of a partial solution (as in Figures 6 and 7) is negligible. The computational cost of CG denoted $T(CG)$ becomes $O(I) + \alpha|S|M + \beta|S|M$. The first term comes from row 1 in Figure 6. The second term comes from rows 3 through 7. The third term comes from row 8. This yields $T(CG) = (1 + \alpha + \beta)O(I)$. Since β is much smaller than α ($\alpha \geq 1$, $\beta < 1$), we can approximate $T(CG)$ with $(1 + \alpha)O(I)$. The key point is that, if α is not too large, we would expect CG to have a running time that is not much longer than that of G.

4 Computational Results

The aim of this section is to verify that CG is able to improve upon the quality of solutions obtained from the greedy algorithm. In particular, if we increase the computational effort required by the greedy algorithm by a factor h, where h is not too large, do we consistently obtain considerably better solutions?

We will see that, in our experiments, for $h \leq 20$, it is possible to achieve this goal. In addition, we will observe how CG can be applied in order to outperform several well–known metaheuristics.

We point out that all software tested in our work and described in this paper was developed in Java 1.7 and executed on a $2.6GHz$ Intel Core i7 machine. Parameter values for α and β have been

obtained by empirical testing.

4.1 Computational Results for the Minimum Label Spanning Tree Problem

There are two parameters, α and β, in the carousel greedy algorithm. Increasing α means increasing the number of basic iterations (i.e., rows in Figure 6 or pictures in Figure 7) and, therefore, the running time. The parameter β is related to each partial solution whose size of $(1 - \beta)|S|$ is maintained throughout α turns of the carousel within CG.

In order to learn more about the interplay of these two parameters, we conducted an experiment in which we applied CG to the $MLST$ problem. In particular, we generated 20 random labeled graphs with 100 nodes, 400 edges, and 100 labels. As before, we know the optimal solution in advance; here it is 25 labels in all 20 cases. The goal was to test different values of α and β. The results are presented in Table 1. For example, when $\alpha = 5$ and $\beta = 10\%$, CG reduces the average gap between the $MVCA$ solution and the optimal solution by 75%.

The best results in Table 1 are indicated in bold. There are several important points to make. First, we see that there are numerous combinations of α and β that reduce the gap by at least 70%. Second, we observe that α and β can be quite small and still produce excellent results.

Suppose we let $\alpha = 5$ and $\beta = 10\%$. Then, if we compare $MVCA$, CG, and the Pilot method [7] on this small set of instances, with respect to average number of labels, we find the following : $MVCA$ has an average of 29.30 labels, the Pilot method has an average of 27.55 labels, and CG has an average of 26.06.

$MVCA$ is a greedy algorithm and it is very fast. On this small set of instances, running times averaged 5.25 milliseconds. The Pilot method required 472 milliseconds, on average. The CG running times are presented in Table 2. As we can see, the running time depends primarily on α, as we anticipated. (As we look down a column in Table 2, the running times increase slightly, rather than decrease slightly, as our analysis from Section 3 would have predicted. This is due to the fact that some iterations of $MVCA$ take longer than others and this is influenced by the value of β.) In Section 4.5, we carried out a more extensive analysis of the parameters α and β on a different problem with more challenging instances and we come to similar conclusions.

Next, we conducted an experiment to see if CG is capable of mitigating the limitation of $MVCA$

α	β				**avg**
	5%	**10%**	**15%**	**20%**	
1	60%	55%	54%	38%	52%
2	72%	71%	60%	47%	63%
3	72%	73%	60%	41%	62%
4	**75%**	72%	66%	47%	65%
5	74%	**75%**	67%	52%	67%
6	**75%**	74%	62%	55%	67%
7	73%	71%	62%	44%	63%
8	73%	72%	64%	55%	66%
avg	72%	70%	62%	47%	

Table 1: CG improvements over $MVCA$

α	β				**avg**
	5%	**10%**	**15%**	**20%**	
1	11	10	10	10	10
2	17	16	15	15	16
3	24	22	21	21	22
4	29	27	27	27	28
5	37	35	34	32	35
6	42	39	38	38	39
7	49	46	43	43	45
8	54	54	50	51	52
avg	33	31	30	30	

Table 2: CG running time in milliseconds

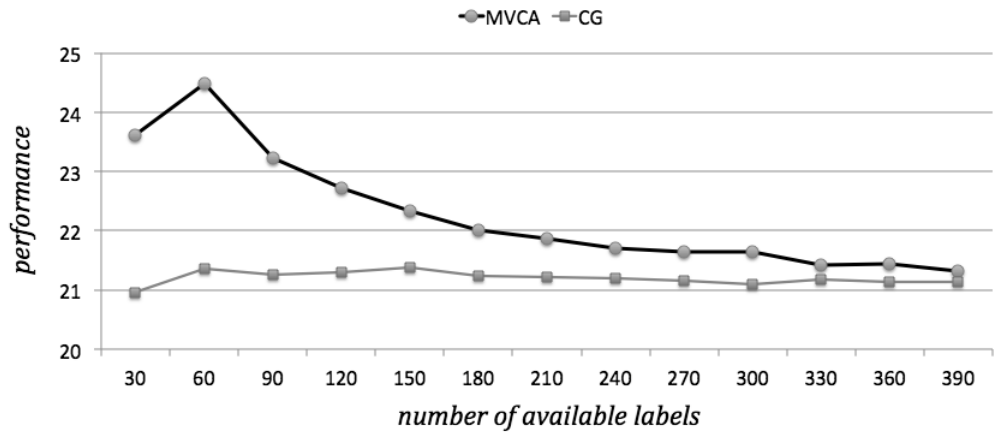

Figure 8: The graph indicates the average number of labels in the MVCA solution, with respect to the number of available labels (20 is the optimal solution for all the instances). The CG performance is not affected by the available number of labels, in contrast to what happens with $MVCA$.

illustrated in Figure 5. We set $\alpha = 5$ and $\beta = 10\%$. The results of this experiment are presented in Figure 8. They indicate that CG can produce excellent solutions for all values of $|L|$.

Since the first step of CG for the $MLST$ problem is to obtain the $MVCA$ solution, this means that in Table 1 and Figure 8, we have a direct comparison between the CG final solution and its initial solution.

The experiments reported on in Tables 1 and 2 are preliminary and involve only 20 instances. A more comprehensive study is presented in Tables 3 and 4. In Table 3 we focus on "small" instances. By this, we mean instances in which the number of labels in the optimal solution is relatively small. In Table 4, we solve "larger" instances.

In Table 3, 20 instances are generated for each row. The numbers reported are averages. The graphs are Group II graphs from Xiong et al. [35]. They are constructed in order to challenge the selection strategy of $MVCA$. We point out that n is the number of nodes, ℓ is the number of labels, b is the number of edges with the same label, and "Size" is the cardinality of the solution. Best results are reported in bold.

In Table 3, CG is compared with $MVCA$, iterated greedy, and Pilot. Iterated greedy repeats Steps 2 & 3 until 50 iterations with no improvement are achieved. In Step 2, 20% of the labels are removed randomly and, in Step 3, $MVCA$ is applied to add labels (see Appendix D). The best feasible solution obtained is reported. CG was run with $\alpha = 5$ and $\beta = 10\%$ for all instances. (These parameter values were used in Table 4 also.)

In Table 3, CG outperforms $MVCA$ in every row. CG and iterated greedy each produce the best solution in seven rows. CG is faster than iterated greedy which is much faster than Pilot. On the other hand, Pilot produces the best solution in 12 rows. When we look at the individual instances, CG produces 225 best solutions and iterated greedy produces 251 best solutions. However, if we focus on the larger instances (with 100, 150, and 200 nodes), CG produces more best solutions than iterated greedy (130 to 106). As instances size increases, CG outperforms iterated greedy and it is computationally less expensive.

Instance		Optimal	MVCA		Carousel		Pilot		Iterated Greedy	
n, ℓ	b	Size	Size	Time	Size	Time	Size	Time	Size	Time
50	4	13	14.40	0.75	**13.00**	4.95	13.35	31.55	**13.00**	8.85
50	5	10	11.65	0.80	11.05	4.20	10.90	28.80	**10.80**	7.10
50	10	5	6.30	0.35	6.10	3.40	**5.80**	19.65	5.90	8.00
50	15	4	4.50	0.20	4.30	3.05	**4.00**	15.90	**4.00**	10.40
50	20	3	3.65	0.55	3.90	2.80	**3.00**	13.40	**3.00**	13.05
75	4	19	21.80	2.40	19.75	13.85	20.40	152.40	**19.10**	36.75
75	7	11	13.25	2.00	**11.90**	11.89	12.00	117.35	11.95	17.65
75	15	5	6.70	1.25	6.30	9.35	**5.65**	82.70	6.00	25.50
75	25	3	4.30	1.10	4.30	7.15	**3.00**	57.40	3.95	23.95
100	4	25	29.30	5.25	26.00	31.15	27.55	472.25	**25.95**	79.95
100	10	10	12.85	3.60	**11.55**	23.40	11.85	310.75	11.75	39.85
100	20	5	7.05	2.60	6.60	18.05	**5.85**	219.50	6.60	43.05
100	30	4	5.05	2.10	4.90	15.45	**4.00**	168.25	4.85	42.55
150	4	38	43.90	17.40	**38.85**	108.10	42.20	2,513.35	**38.85**	341.60
150	15	10	13.20	9.30	**12.05**	62.75	**12.05**	1,269.40	12.40	94.60
150	30	5	7.30	6.65	6.95	46.10	**6.05**	901.85	7.00	112.70
150	45	4	5.15	5.65	5.05	40.20	**4.00**	700.80	4.95	123.90
200	4	50	59.00	41.65	**51.50**	260.20	56.20	7,963.50	52.05	964.95
200	20	10	13.90	18.85	**12.95**	131.25	13.00	3,474.40	13.20	207.15
200	40	5	7.75	13.55	7.10	98.55	**5.80**	2,515.55	7.35	214.30
200	60	4	5.75	12.70	5.10	92.15	**4.00**	2,439.45	5.05	307.65
# best solutions			**52**		**225**		**310**		**251**	

Table 3: $MLST$ results for "small" instances (time is shown in milliseconds)

In Table 4, the number of labels in the optimal solution is larger and there are five instances per row. Pilot is not able to produce the best solution for any row and its running times are large. It is clear for this table that CG obtains better solutions than iterated greedy and it is considerably faster.

CG, Pilot, and iterated greedy are all based on the $MVCA$ approach. Table 3 and 4 reveal that each of the three algorithms outperforms the greedy algorithm. CG is, obviously, the fastest of the three. In addition, Table 4 suggests that, as we increase the size of the problem instance, CG outperforms Pilot and iterated greedy with respect to both accuracy and running time.

In order to compare solution quality in a statistical sense, we use the non–parametric Friedman test [18]. First, we compare the results (size of solution) from carousel greedy, Pilot, and iterated greedy in Table 3. The null hypothesis H_0 is that none of the techniques is better than another. The alternate hypothesis H_1 is that there is a difference in solution quality among the three procedures. We obtain a chi–squared value of 1.4615 with 2 degrees of freedom and a p–value of 0.4815; this means that we cannot reject the null hypothesis.

Turning our attention to Table 4, we find a chi–squared value of 32.2278 with 2 degrees of freedom and a p–value of 1.004×10^{-7}; this means that we can reject the null hypothesis. Given that CG produces the best solutions in 16 out of 20 rows, CG is clearly the winning approach in Table 4.

Taking both solution quality and running time into account, CG outperforms Pilot and iterated greedy for small and large $MLST$ instances.

Instance		Optimal	MVCA		Carousel		Pilot		Iterated Greedy	
n, ℓ	b	Size	Size	Time	Size	Time	Size	Time	Size	Time
100	4	25	29.30	0.01	26.00	0.03	27.55	0.47	**25.95**	0.08
100	8	13	15.40	0.00	14.00	0.03	14.00	0.35	**13.60**	0.08
200	4	50	57.00	0.04	**51.80**	0.26	55.20	7.96	52.40	0.90
200	8	25	31.20	0.03	28.00	0.20	28.60	5.66	**27.80**	0.61
300	4	75	88.60	0.14	**77.00**	0.91	82.80	40.37	79.40	3.11
300	8	38	46.60	0.10	42.00	0.67	43.60	27.15	**40.80**	2.84
400	4	100	117.60	0.35	**102.40**	2.16	111.40	127.78	106.00	8.56
400	8	50	61.40	0.23	**55.40**	1.58	58.60	86.13	55.80	5.71
500	4	125	147.80	0.71	**129.20**	4.56	140.60	319.54	133.00	18.02
500	8	63	78.60	0.49	**69.20**	3.48	73.60	218.39	70.20	14.19
600	4	150	177.60	1.26	**154.60**	8.29	169.20	681.54	159.60	32.51
600	8	75	91.60	0.83	**82.80**	6.16	88.40	470.67	85.00	24.85
700	4	175	205.60	2.05	**180.40**	13.72	197.20	1,380.29	187.00	57.70
700	8	88	108.60	1.46	**98.00**	10.92	103.40	956.78	98.80	46.52
800	4	200	234.60	3.21	**206.80**	21.84	225.80	2,565.06	214.40	88.55
800	8	100	123.60	2.28	**110.40**	17.11	118.60	1,750.07	113.40	91.25
900	4	225	263.40	4.78	**232.20**	33.06	255.20	4,398.87	241.00	150.16
900	8	113	140.20	3.52	**126.00**	26.07	133.60	2,964.55	127.20	129.13
1000	4	250	293.20	6.94	**258.00**	47.83	284.00	6,988.42	269.60	214.23
1000	8	125	155.80	5.05	**140.00**	37.29	148.80	4,457.36	141.80	159.18
# best solutions			**0**		**86**		**3**		**30**	

Table 4: $MLST$ results for "large" instances (time is shown in seconds)

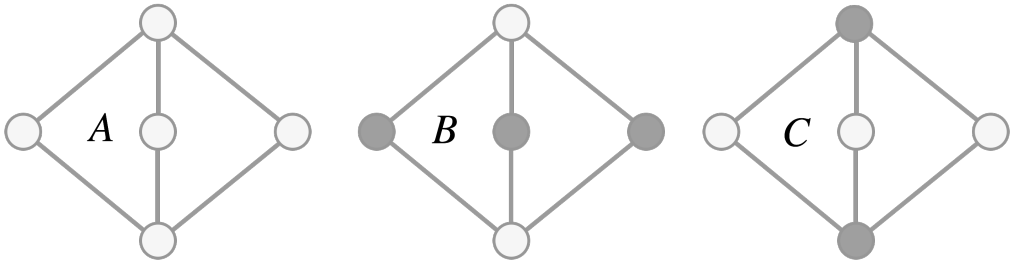

Figure 9: Given the input graph in A, B shows a vertex cover with 3 (dark) nodes and C shows the minimum vertex cover with 2 (dark) nodes

4.2 Computational Results for the Minimum Vertex Cover Problem

Another important problem in combinatorial optimization is the minimum vertex cover (MVC) problem. In graph theory, a vertex (node) cover of a graph is a set of nodes such that each edge of the graph is incident to at least one node of the set. The MVC problem is to find the vertex cover of minimum cardinality (see Figure 9 and Appendix G). This is one of Karp's 21 NP-complete problems (see Karp [27]). We compare CG with a greedy algorithm for the MVC problem which we denote by Greedy. This algorithm selects the node with the largest degree, removes the edges incident to that node, and repeats until all edges in the graph have been removed. This greedy algorithm was designed by Chvatal 1979 [9] for the set covering problem which generalizes the vertex cover problem. Later on, it was used for the vertex cover problem by Hochbaum 1983 [23], and Clarkson 1983 [10].

In Table 5, we report results for the BHOSLIB graphs [28]. The BHOSLIB benchmark instances were designed to be especially difficult MVC instances. There is one instance per row. For both Tables 5 and 6, we set $\alpha = 20$ and $\beta = 1\%$.

From Table 5A, the gap between the Greedy results and optimality is decreased by 67%, on average, by CG. For this data set, the greedy algorithm is, again, very fast. CG is able to improve upon greedy in all rows. We set $\alpha = 20$ since the running times were so small, although Table 10 indicates that we could have used $\alpha = 10$ instead.

In Table 6A, we apply Greedy and CG to MVC instances from the DIMACS graphs [25]. These graphs were originally designed for the maximum clique problem. In Table 6A, when we refer to a specific graph G from the DIMACS library, we mean the complement graph $\overline{G}$ (see Harary [21] for a formal definition). In this table, CG reduces the gap between the Greedy results and optimality by about 37%, on average. CG produces the optimal solution for 25 instances whereas Greedy finds the optimal solution for 15 instances. In Tables 5A and 6A, the running times are proportional to the parameter α, as expected; with $\alpha = 20$, CG running times are about 20 times as long as greedy running times.

4.3 Computational Results for the Maximum Independent Set Problem

An independent set in graph theory is a subset of nodes in the graph such that no two of them are adjacent. In Figure 9C, the dark nodes are an independent set of size 2. In Figure 9B, the dark nodes form the maximum independent set of size 3. The maximum independent set (MIS) problem is to find an independent set of the largest possible size (see Appendix G). The MIS problem is one of the classical NP-complete problems [27] and it is closely related to the maximal clique and minimum vertex cover problems.

A greedy algorithm for the MIS problem, based on the work of Chvatal [9], is as follows: Find a node with the minimum degree, remove it and its neighbors from the current graph, and add it to the solution. Continue until the current graph is empty. We will refer to this procedure as Greedy.

As in Section 4.2, we apply Greedy and CG to the BHOSLIB and DIMACS instances. Since we are solving a maximization problem here, we refer the reader to Appendix B (maximization form) for the relevant CG pseudo–code. (We point out that the minimization form and maximization form differ only in line 7.) We set $\alpha = 20$ and $\beta = 1\%$.

In Table 5B, we focus on the BHOSLIB graphs. The gap between optimality and the Greedy results is decreased by 36%, on average, by CG. In Table 6B, we study the DIMACS graphs. The gap between optimality and Greedy results is decreased by about 16%, on average, by CG.

With respect to running times in Tables 5B and 6B, we note that they are not proportional to α. In fact, they are much smaller than expected. The explanation is as follows: The greedy algorithm selects nodes and removes edges from one iteration to the next. After it has selected a majority of the nodes that it will select, the remaining graph is small and the work per iteration is substantially reduced. CG operates in this same environment; each iteration is inexpensive. For this reason, CG running times are only slightly longer than the Greedy running times.

	A						B					
Instance	**Minimum Vertex Cover**						**Maximum Independent Set**					
	Opt.	**Greedy**		**Carousel**		**Gap**	**Opt.**	**Greedy**		**Carousel**		**Gap**
	Size	**Size**	**ms**	**Size**	**ms**		**Size**	**Size**	**ms**	**Size**	**ms**	
frb30-15-1	**420**	429	3	424	70	56%	**30**	25	3	26	8	20%
frb30-15-2	**420**	431	4	422	76	82%	**30**	24	3	28	8	67%
frb30-15-3	**420**	429	3	423	50	67%	**30**	24	2	26	6	33%
frb30-15-4	**420**	429	3	424	46	56%	**30**	25	3	28	6	60%
frb30-15-5	**420**	430	3	423	62	70%	**30**	26	2	27	5	25%
frb35-17-1	**560**	570	5	563	90	70%	**35**	28	6	30	12	29%
frb35-17-2	**560**	570	6	564	90	60%	**35**	30	6	32	10	40%
frb35-17-3	**560**	571	5	563	92	73%	**35**	29	5	32	10	50%
frb35-17-4	**560**	570	4	564	88	60%	**35**	28	6	32	12	57%
frb35-17-5	**560**	571	4	564	95	64%	**35**	30	5	31	10	20%
frb40-19-1	**720**	737	9	724	175	76%	**40**	33	11	35	17	29%
frb40-19-2	**720**	732	9	725	165	58%	**40**	33	11	34	18	14%
frb40-19-3	**720**	733	9	724	171	69%	**40**	34	10	36	17	33%
frb40-19-4	**720**	732	9	723	168	75%	**40**	32	10	36	20	50%
frb40-19-5	**720**	733	9	725	171	62%	**40**	33	10	36	18	43%
frb45-21-1	**900**	913	14	906	278	54%	**45**	37	20	39	31	25%
frb45-21-2	**900**	916	14	904	275	75%	**45**	35	18	39	35	40%
frb45-21-3	**900**	917	14	905	271	71%	**45**	35	16	39	31	40%
frb45-21-4	**900**	913	14	906	271	54%	**45**	38	18	41	28	43%
frb45-21-5	**900**	917	14	905	273	71%	**45**	37	18	41	31	50%
frb50-23-1	**1100**	1118	21	1105	401	72%	**50**	42	27	44	38	25%
frb50-23-2	**1100**	1119	20	1105	403	74%	**50**	41	27	44	42	33%
frb50-23-3	**1100**	1119	20	1107	399	63%	**50**	40	29	44	50	40%
frb50-23-4	**1100**	1116	20	1106	398	63%	**50**	40	39	43	61	30%
frb50-23-5	**1100**	1118	20	1106	409	67%	**50**	42	29	45	44	38%
frb53-24-1	**1219**	1240	24	1226	496	67%	**53**	44	38	47	59	33%
frb53-24-2	**1219**	1240	24	1224	509	76%	**53**	44	36	48	59	44%
frb53-24-3	**1219**	1235	25	1224	493	69%	**53**	46	35	48	48	29%
frb53-24-4	**1219**	1237	26	1226	513	61%	**53**	44	39	47	59	33%
frb53-24-5	**1219**	1239	26	1226	518	65%	**53**	41	32	46	56	42%
frb56-25-1	**1344**	1367	30	1350	599	74%	**56**	46	45	49	73	30%
frb56-25-2	**1344**	1365	32	1350	603	71%	**56**	45	46	50	72	45%
frb56-25-3	**1344**	1363	30	1352	595	58%	**56**	46	48	50	75	40%
frb56-25-4	**1344**	1363	31	1350	604	68%	**56**	45	44	48	70	27%
frb56-25-5	**1344**	1366	30	1351	599	68%	**56**	47	41	49	57	22%
frb59-26-1	**1475**	1494	36	1482	715	63%	**59**	47	51	50	82	25%
frb59-26-2	**1475**	1496	36	1483	729	62%	**59**	49	56	52	84	30%
frb59-26-3	**1475**	1501	37	1481	732	77%	**59**	48	54	52	84	36%
frb59-26-4	**1475**	1496	36	1482	714	67%	**59**	49	46	55	90	60%
frb59-26-5	**1475**	1494	36	1483	737	58%	**59**	49	49	51	70	20%
Average values						**67%**						**36%**

Table 5: Minimum vertex cover problem and maximum independent set problem results for BHOSLIB graphs (ms represents the running time in milliseconds)

Instance	A: Minimum Vertex Cover						B: Maximum Independent Set					
	Opt.	Greedy		Carousel		Gap	Opt.	Greedy		Carousel		Gap
	Size	Size	ms	Size	ms		Size	Size	ms	Size	ms	
brock200_1	**179**	184	4	181	23	60%	**21**	20	2	20	5	0%
brock200_2	**188**	192	4	191	58	25%	**12**	9	1	9	5	0%
brock200_3	**185**	189	1	187	20	50%	**15**	12	1	13	6	33%
brock200_4	**183**	188	1	186	13	40%	**17**	13	0	14	1	25%
brock400_1	**373**	382	4	377	52	56%	**27**	21	3	23	6	33%
brock400_2	**371**	379	3	377	53	25%	**29**	21	2	23	5	25%
brock400_3	**369**	383	3	376	49	50%	**31**	20	2	22	4	18%
brock400_4	**367**	380	3	376	53	31%	**33**	22	3	23	5	9%
brock800_1	**777**	786	27	781	566	56%	**23**	18	10	19	19	20%
brock800_2	**776**	788	26	782	557	50%	**24**	18	12	19	18	17%
brock800_3	**775**	786	27	781	565	45%	**25**	17	11	19	18	25%
brock800_4	**774**	787	27	781	575	46%	**26**	17	11	18	17	11%
c-fat200-1	**188**	**188**	1	**188**	19	0%	**12**	**12**	1	**12**	2	0%
c-fat200-2	**176**	**176**	1	**176**	18	0%	**24**	**24**	1	**24**	3	0%
c-fat200-5	**142**	**142**	1	**142**	10	0%	**58**	**58**	1	**58**	7	0%
c-fat500-1	**486**	**486**	24	**486**	578	0%	**14**	**14**	4	**14**	8	0%
c-fat500-10	**374**	**374**	12	**374**	262	0%	**126**	**126**	37	**126**	95	0%
c-fat500-2	**474**	**474**	17	**474**	398	0%	**26**	**26**	6	**26**	14	0%
c-fat500-5	**436**	**436**	16	**436**	348	0%	**64**	**64**	16	**64**	60	0%
DSJC1000_5	**985**	990	63	987	1355	60%	**15**	13	18	13	24	0%
gen200_p0.9_44	**156**	169	1	161	9	62%	**44**	35	0	**44**	2	100%
gen200_p0.9_55	**145**	165	1	**145**	6	100%	**55**	39	0	41	1	13%
gen400_p0.9_55	**345**	371	2	348	26	88%	**55**	45	2	50	6	50%
gen400_p0.9_65	**335**	368	2	352	26	48%	**65**	46	2	47	5	5%
gen400_p0.9_75	**325**	363	1	350	26	34%	**75**	44	2	50	7	19%
hamming10-2	**512**	**512**	4	**512**	81	0%	**512**	**512**	23	**512**	58	0%
hamming10-4	**984**	992	25	988	504	50%	**40**	36	17	36	23	0%
hamming6-2	**32**	**32**	0	**32**	1	0%	**32**	**32**	0	**32**	0	0%
hamming6-4	**60**	**60**	0	**60**	0	0%	**4**	**4**	0	**4**	0	0%
hamming8-2	**128**	**128**	1	**128**	7	0%	**128**	**128**	1	**128**	7	0%
hamming8-4	**240**	**240**	1	**240**	18	0%	**16**	**16**	0	**16**	1	0%
johnson16-2-4	**112**	**112**	0	**112**	2	0%	**8**	**8**	0	**8**	0	0%
johnson32-2-4	**480**	**480**	3	**480**	48	0%	**16**	**16**	1	**16**	2	0%
johnson8-2-4	**24**	**24**	0	**24**	0	0%	**4**	**4**	0	**4**	0	0%
johnson8-4-4	**56**	62	0	**56**	1	100%	**14**	**14**	0	**14**	0	0%
keller4	**160**	163	1	**160**	6	100%	**11**	**11**	0	**11**	1	0%
keller5	**749**	761	18	754	371	58%	**27**	21	12	23	19	33%
MANN_a27	**252**	261	1	255	14	67%	**126**	125	1	125	6	0%
MANN_a45	**690**	705	6	705	96	0%	**345**	342	10	342	34	0%
MANN_a9	**29**	33	0	**29**	0	100%	**16**	**16**	0	**16**	0	0%
p_hat300-1	**292**	293	4	293	74	0%	**8**	7	1	7	2	0%
p_hat300-2	**275**	278	3	**275**	43	100%	**25**	**25**	2	**25**	3	0%
p_hat300-3	**264**	269	1	**264**	26	100%	**36**	33	1	**36**	3	100%
p_hat500-1	**491**	493	16	**491**	346	100%	**9**	8	5	8	8	0%
p_hat500-2	**464**	468	11	**464**	236	100%	**36**	34	8	**36**	15	100%
p_hat700-1	**689**	693	32	690	814	75%	**11**	8	9	9	13	33%
san1000	**985**	992	60	992	1525	0%	**15**	9	26	9	42	0%
san200_0.7_1	**170**	185	1	185	10	0%	**30**	16	1	16	2	0%
san200_0.7_2	**182**	188	1	188	10	0%	**18**	15	0	15	1	0%
san200_0.9_1	**130**	155	1	154	6	4%	**70**	47	0	47	2	0%
san200_0.9_2	**140**	165	0	160	5	20%	**60**	39	0	41	1	10%
san200_0.9_3	**156**	176	0	165	5	55%	**44**	34	0	35	1	10%
san400_0.5_1	**387**	393	8	393	152	0%	**13**	9	2	9	4	0%
san400_0.7_1	**360**	380	4	380	68	0%	**40**	21	3	21	6	0%
san400_0.7_2	**370**	385	4	385	65	0%	**30**	17	2	18	4	8%
san400_0.7_3	**378**	388	3	387	67	10%	**22**	14	2	16	5	25%
san400_0.9_1	**300**	350	1	350	26	0%	**100**	80	2	**100**	19	100%
sanr200_0.7	**182**	184	1	**182**	8	100%	**18**	16	0	17	1	50%
sanr200_0.9	**158**	164	0	159	5	83%	**42**	37	1	41	2	80%
sanr400_0.5	**386**	392	7	390	121	33%	**14**	12	2	12	3	0%
sanr400_0.7	**379**	384	4	**379**	58	100%	**21**	19	2	20	4	50%
Average values						**37%**						**16%**

Table 6: Minimum vertex cover problem and maximum independent set problem results for DIMACS graphs (ms represents the running time in milliseconds)

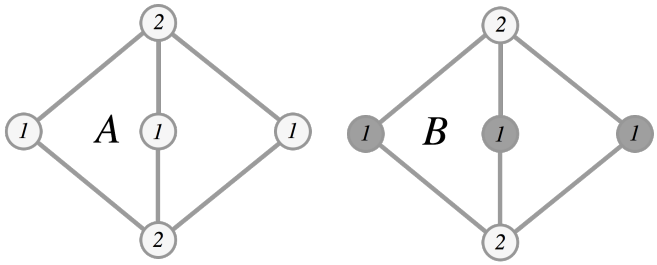

Figure 10: Given the input graph in A, with node weights shown, the $MWVC$ solution is indicated by the dark nodes in B

4.4 Computational Results for the Minimum Weight Vertex Cover Problem

So far, we have applied CG to problems in which we seek to minimize or maximize the cardinality of a set. We have focused on this class of problems because we believe that the application of CG in this setting is natural and intuitive (it corresponds to adding or deleting an element in the carousel). In this section, however, we show how CG can be applied to more complex combinatorial optimization problems. For example, suppose we seek to minimize or maximize a weighted sum of selected elements.

In particular, we consider the minimum weight vertex cover ($MWVC$) problem. This is a straightforward generalization of the MVC problem where each node has a positive weight. The $MWVC$ problem is to find the vertex cover that minimizes the sum of the weights associated with the selected nodes (see Figure 10 and Appendix G). The $MWVC$ problem is NP-hard since the MVC problem is a special case in which each node has a weight of one.

The basic greedy algorithm (Greedy) is again derived from Chvatal [9]. The idea is to add the node v with the minimum ratio $\frac{w(v)}{d(v)}$ to the solution at each step, where $w(v)$ is the weight and $d(v)$ is the degree of node v. Then, all edges incident to that node are removed from the graph and the process continues until no edges remain.

The relevant CG pseudo–code is provided in Appendix C. In our experiments, we set $\alpha = 20$ and $\beta = 5\%$.

In Table 8, we compare Greedy and CG with several metaheuristic algorithms developed and tested in Shyu et al. [32]. In particular, Shyu et al. compared simulated annealing (SA), tabu search (TS), and ant colony optimization (ACO) on moderate-sized problems in Tables 4 and 5 of their paper.

There are several recent papers [3, 26, 33, 34, 37] on the $MWVC$ problem. These papers introduce new metaheuristics that improve upon the results in [32] by a small amount. The running times are comparable to the ACO running times in [32]. The average gap between ACO and the now best-known solutions is less than the 0.88% .

More specifically, they considered 39 combinations of number of nodes and number of edges. For each combination, they generated 10 random instances. We followed their protocol and added columns for Greedy (abbreviated $GRDY$) and CG. We used the same random graphs tested in their paper. The results from Table 8 show that CG outperformed SA and TS, but not ACO. Despite the fact that $\alpha = 20$, CG required only about eight times the running time of Greedy. For this experiment

Nodes	Edges	GRDY	CG	SA	TS	ACO
50	50	0	1	183	175	63
	100	0	1	219	212	83
	250	0	1	272	266	97
	500	0	1	320	319	102
	750	0	1	359	358	125
	1000	0	1	370	358	117
300	300	2	13	5794	5527	4322
	500	2	15	6539	6527	5178
	750	4	19	7067	6955	6055
	1000	2	18	7391	7222	6231
	2000	3	21	8048	8409	6488
	3000	5	25	8372	8995	6299
	5000	7	30	8762	9122	6558

Table 7: $MWVC$ problem running times in milliseconds

we implement the greedy algorithm and CG; for the other algorithms, we report the results from [32]. Unlike $GRDY$ and CG, SA, TS, and ACO were developed in $C++$ and executed on a $1.7GHz$ AMD machine. Nevertheless, we report the running times for the smallest and the largest sets from the six sets of instances (from Table 8) in Table 7, in order to contrast running times. The key observation is that the three metaheuristic approaches (SA, TS, and ACO) require hundreds of milliseconds for every one required by CG.

Nodes	Edges	GRDY	CG	SA	TS	ACO
50	50	1343.7	1320.5	1339.4	1339.4	1282.1
	100	1858.3	1801.4	1829.6	1826.8	1741.1
	250	2416	2329.3	2398.2	2398.2	2287.4
	500	2800.8	2742.9	**2741.7**	2760.8	2679
	750	3052.8	3013.2	3032	3032	2959
	1000	3234.7	3229.3	3234.7	3234.7	3211.2
100	100	2676.7	2653.8	2663.2	2668.3	2552.9
	250	3816.7	3727.4	3765.4	3765.4	3626.4
	500	4906.8	4773.6	4791.8	4793.3	4692.1
	750	5282.7	5192.8	5227.9	5221.7	5076.4
	1000	5749.9	5594.6	5665	5649.1	5534.1
	2000	6184.5	6141.9	6159.5	6170.5	6095.7
150	150	3864.1	3808.4	3834.2	3837.4	3684.9
	250	5019	4926.7	4953.3	4951.8	4769.7
	500	6551.6	6357.5	6449	6452.5	6224
	750	7291.3	7131.1	7220.3	7222.4	7014.7
	1000	7747.7	7536.3	7623.9	7614.8	7441.8
	2000	8900.8	8744.5	8767.3	8767.6	8631.2
	3000	9178.9	9033.9	9070.2	9071.7	8950.2
200	250	5892.9	5808.7	5841.7	5841.7	5588.7
	500	7620.8	7444.8	7521.4	7518.6	7259.2
	750	8721.2	8466.7	8590.9	8604.9	8349.8
	1000	9729.2	9432.5	9537.6	9507.7	9262.2
	2000	11341	11042.1	11076.8	11076.6	10916.5
	3000	12027.5	11803.8	11829.5	11834.7	11689.1
250	250	6508.3	6421.5	6475.8	6475	6197.8
	500	8960.3	8796.5	8874.9	8869.4	8538.8
	750	10342.4	10125.5	10205.6	10211.8	9869.4
	1000	11313	11083.9	11183.4	11165.5	10866.6
	2000	13361.7	13081.9	13110.3	13097.4	12917.7
	3000	14320	14032.7	14058	14056.9	13882.5
	5000	15045.9	14830.5	14957.8	14940.1	14801.8
300	300	7695.6	7579.2	7634.9	7634	7342.7
	500	10011.7	9750.5	9844.5	9844.4	9517.4
	750	11700.6	11407.9	11570.5	11565.6	11166.9
	1000	12825.5	12517.5	12618	12618.7	12241.7
	2000	15467.4	15091.3	15211.5	15201.5	14894.9
	3000	16495.4	16208.7	16265.5	16243.2	16054.1
	5000	17935	17563.4	17681.6	17648	17545.4
Percentage above ACO		**4.13%**	**1.99%**	**2.82%**	**2.81%**	**0.00%**

Table 8: Computational results for the $MWVC$ problem (ACO obtains the best results for all the instances and CG does second best on all instances except for the one in bold)

4.5 Setting α and β

The meaning of the parameters α and β is widely discussed in the previous sections. In Table 9, we compile the settings of α and β for each of the four different optimization problems discussed so far. For each of these problems, we experimentally identified the best parameter values.

Based on Tables 1 and 2 with respect to the $MLST$ problem and our experience with the MVC, MIS, and $MWVC$ problems, we observe the following:

(i) As α increases from 1, running time increases and so does solution quality, until it stabilizes;

(ii) The parameter β is related primarily to solution quality.

In order to learn more about the interaction of α and β, we designed a small experiment using the MVC problem. In particular, we focused on the five instances $frb30-15$ with optimal solution values of 420. In Table 10, we see average solution size for various settings of α and β. In Table 11, we see the associated running times.

Problem	α	β
$MLST$	5	10%
MVC	20	1%
MIS	20	1%
$MWVC$	20	5%

Table 9: The settings of α and β in the previous sections

Figures 11 to 14 are drawn from the data in Tables 10 and 11. Figure 11 reveals that increasing α up to 10 improves the solution quality. Beyond $\alpha = 10$, solution quality does not improve. Figure 12 shows that a value of β between 0.5% and 2.5% produces the highest quality solutions. Figure 13 illustrates the direct connection between α and running time. In Figure 14, we see that increasing β has a small impact on running time.

Based on our experience with CG to date, we would recommend that users set α between 5 and 20 and β between 1% and 10%.

It is possible for CG to remove and later select the same element. If this happens for a full turn of the carousel at iteration i, then the remaining $\alpha - i$ iterations are unnecessary. In this case, we say that CG produces a loop or a cycle. In future work, we plan to investigate ways to quickly detect and avoid loops.

5 A Hybrid Carousel Greedy Algorithm

So far, we have shown that CG is a versatile approach to amplify the power of greedy algorithms. In some of our experiments, we have even shown that CG alone can outperform some metaheuristics (see Tables 4 and 8). In this section, we demonstrate that CG can be combined with a traditional metaheuristic to create a powerful Hybrid CG algorithm. This can be accomplished in a number of ways. However, in this paper, we demonstrate the possibilities by combining CG with the Pilot method. As an alternative, CG can easily be combined with a genetic algorithm [6, 35], but we leave that for future work.

α	β										avg
	0.5%	**1.0%**	**1.5%**	**2.0%**	**2.5%**	**3.0%**	**3.5%**	**4.0%**	**4.5%**	**5.0%**	
1	427.6	426.4	427.4	426.8	427.6	428.2	428.2	428.0	428.4	428.4	427.7
2	425.6	425.6	424.8	426.2	425.6	427.8	426.0	428.0	426.8	427.2	426.4
3	425.4	425.2	424.6	425.2	426.0	425.6	426.0	426.0	426.6	427.0	425.8
4	424.4	424.8	424.6	425.0	424.8	424.6	425.8	426.2	427.0	426.8	425.4
5	425.2	424.4	424.2	424.6	424.0	425.4	425.8	426.0	426.2	426.6	425.2
6	**423.4**	424.4	424.4	424.4	424.8	424.8	426.4	425.6	426.6	426.0	425.1
7	424.4	**423.8**	**423.8**	424.2	**423.6**	425.4	425.4	426.2	427.0	425.8	425.0
8	**423.2**	**423.4**	424.2	**423.6**	424.6	424.4	425.6	426.6	425.6	426.0	424.7
9	**423.8**	**423.2**	**423.4**	**423.4**	**423.8**	424.6	426.2	426.4	425.6	426.4	424.7
10	**423.8**	**423.4**	**423.2**	**423.2**	**423.6**	424.4	425.2	426.0	425.4	425.2	424.3
11	**423.6**	**423.0**	**422.8**	**423.8**	424.2	425.2	424.6	425.2	424.8	426.0	424.3
12	**422.8**	**423.6**	**423.2**	**423.0**	**423.4**	424.0	426.0	425.4	425.8	425.6	424.3
13	**423.2**	**422.8**	**423.0**	**423.0**	**423.6**	424.0	425.2	425.2	425.4	425.8	424.1
14	**423.4**	**423.8**	**423.6**	**423.2**	424.6	424.0	424.8	426.4	426.0	426.6	424.6
15	**423.4**	**423.4**	**423.0**	**423.2**	**423.6**	**423.6**	424.8	425.4	425.2	426.2	424.2
16	**423.4**	**423.2**	**423.2**	**422.8**	**423.4**	**423.0**	425.2	424.6	426.0	426.0	424.1
17	**423.0**	**423.2**	**423.6**	**423.0**	**423.8**	**423.6**	424.0	426.0	425.4	426.0	424.2
18	**423.2**	**423.2**	**423.2**	**422.8**	**423.6**	424.2	424.8	426.6	425.6	427.0	424.4
19	**423.2**	**423.4**	424.4	**423.0**	**422.8**	424.0	424.8	425.6	425.4	426.2	424.3
20	**423.4**	**422.8**	**423.8**	**423.6**	**423.2**	424.6	424.8	425.8	425.4	426.0	424.3
avg	424.0	423.9	423.9	423.9	424.2	424.8	425.5	426.1	426.0	426.3	

Table 10: CG for the MVC problem: Average solution size for different settings of α and β (in bold are the values that are ≤ 423.8)

The idea is simple. For the sake of convenience, we will again refer to the $MLST$ problem, in order to explain how this Hybrid CG algorithm would work. We borrow from the work of Xiong et al. [36]. To begin, we sort all labels by their frequencies in the input graph G, from highest to lowest. It should be clear that labels that occur most frequently in G are most promising with respect to the $MLST$ problem. Now, suppose L, List 1 in Figure 15 is this sorted list of labels and we apply CG to this $MLST$ instance.

We recall that CG selects a number of labels before arriving at a feasible solution. For example, in Figure 6, CG selected the nine labels B,G,R,Y,P,R,G,Y, and P, in sequence.

In Hybrid CG, we begin with an execution of CG. Each time a label is selected in CG, delete it from List 1. Let's suppose, on a small instance, CG selects labels a,b,d,e,f,h, and i. We, then, delete these labels from List 1 to obtain List 2 (see Figure 15).

We observe that label c is the most promising label on List 2 which CG has not yet encountered. We can, therefore, *restart* CG with label c as a first label. This will diversify our search. we can repeat this for at most k runs of CG (or $k-1$ restarts).

Some initial computational results are presented in Table 12, where the numbers in each row are averages over 20 instances. The Hybrid CG algorithm has three parameters. We set $\alpha = 3$, $\beta = 10\%$, and $k = 10$. We reduced α to 3 (see Table 9), in order to speed up the Hybrid CG algorithm.

We provide pseudo–code for this procedure in Appendix F. For this specific implementation, the input parameter ℓ (depth of search) is equal to one, as in [36].

The Hybrid CG algorithm performs well on the $MLST$ problem instances, as observed in Table 12. First, we can compare Carousel, Hybrid CG, and Pilot in a statistical sense, using the Friedman test. We obtain a chi–squared value of 19 with 2 degrees of freedom and a p–value of 7.485×10^{-5}; this means we can reject the null hypothesis. So, the three procedures do not share the same solution

α	β										avg
	0.5%	**1.0%**	**1.5%**	**2.0%**	**2.5%**	**3.0%**	**3.5%**	**4.0%**	**4.5%**	**5.0%**	
1	59.1	58.8	57.8	57.4	58.0	57.6	56.8	57.4	56.4	57.6	57.7
2	59.2	61.4	61.0	62.0	64.4	64.8	66.4	65.6	65.0	66.4	63.6
3	67.2	67.4	70.0	70.0	71.6	71.6	73.4	74.6	73.8	76.2	71.6
4	73.0	73.0	77.0	77.4	76.2	77.4	83.4	81.6	83.0	83.6	78.6
5	78.2	79.2	81.4	82.4	85.0	86.0	87.6	88.6	91.0	92.4	85.2
6	88.0	88.8	88.0	89.4	91.8	93.0	94.8	98.2	98.2	102.4	93.3
7	90.4	92.8	93.8	96.2	99.2	103.8	103.6	105.2	108.8	110.4	100.4
8	96.6	100.0	99.6	104.0	106.6	109.2	111.0	116.4	116.8	120.8	108.1
9	103.8	99.6	111.2	112.2	115.4	114.4	121.0	124.0	125.6	128.0	115.5
10	109.8	110.6	115.6	118.4	117.4	127.0	128.0	132.4	134.4	136.0	123.0
11	114.4	116.6	121.6	123.4	127.6	130.4	134.6	141.4	142.8	147.6	130.0
12	128.6	127.6	131.2	132.4	136.0	139.6	144.0	149.0	149.2	154.0	139.2
13	128.4	131.2	136.8	137.4	142.6	143.8	150.0	155.2	155.4	163.2	144.4
14	135.2	134.8	140.6	144.2	150.4	158.4	149.8	167.4	167.2	171.2	151.9
15	139.8	144.4	146.6	151.8	155.8	160.8	167.0	172.2	176.2	182.0	159.7
16	151.2	155.8	153.4	160.6	164.2	170.6	173.6	186.2	192.2	192.4	170.0
17	157.8	159.6	163.4	164.4	162.2	172.4	188.6	191.2	194.8	203.6	175.8
18	163.8	164.2	166.4	174.6	180.4	181.8	195.4	201.0	205.6	208.2	184.1
19	166.4	173.6	173.4	183.0	188.6	194.8	196.0	204.8	214.0	217.2	191.2
20	174.0	178.2	184.6	185.6	199.0	204.6	210.2	217.4	225.4	233.0	201.2
avg	114.2	115.9	118.7	121.3	124.6	128.1	131.8	136.5	138.8	142.3	

Table 11: CG for the MVC problem: Running times in milliseconds for different settings of α and β

quality.

Upon examination of Table 12, we see that Hybrid CG is much faster than the Pilot method and it clearly outperforms the Pilot method. More specifically, Hybrid CG outperforms the Pilot method in 12 of the 21 rows. Pilot outperforms Hybrid CG in only two rows and they tie in seven rows.

In Tables 8 and 7, we saw that CG outperformed two metaheuristics (SA and TS) even though its running time was much smaller. Although ACO outperformed CG with respect to quality of solution, CG was hundreds of times faster than ACO. In Table 12, we learned that Hybrid CG was able to outperform Pilot while taking a fraction of the time required by Pilot. Next, we ask the question: Can Hybrid CG outperform ACO ?

Since Tables 8 and 7 involve the $MWVC$ problem, we need to focus on implementing Hybrid CG to solve this problem. For the $MWVC$ problem, List 1 is a list of vertices (instead of labels or colors). Otherwise, Hybrid CG is, essentially, the same as it is for the $MLST$ problem. We start with the CG implementation proposed in Section 4.4, with $\alpha = 20$ and $\beta = 5\%$, and we again set $k = 10$. For the $MWVC$ problem, however, we apply Hybrid CG to a depth of up to l levels (rather than just one).

After the first level, we fix the first node in our solution. After the second level, we fix the second node, and so on, until a depth of l. See [15] for details. We set l equal to half of the $GRDY$ solution.

In Table 13, we compare ACO and Hybrid CG. The running times are shown in milliseconds and, in bold, we report the best solution per row. As previously stated, ACO and Hybrid CG were not developed in the same programming language and the experimental tests were conducted on different machines. Nevertheless, upon examination of Table 13, we see that running times for Hybrid CG are less than half the ACO running times. In a practical sense, Hybrid CG seems to outperform ACO with respect to solution quality. More specifically, Hybrid CG outperforms ACO in 24 of the 39 rows; ACO outperforms Hybrid CG in only 15 rows.

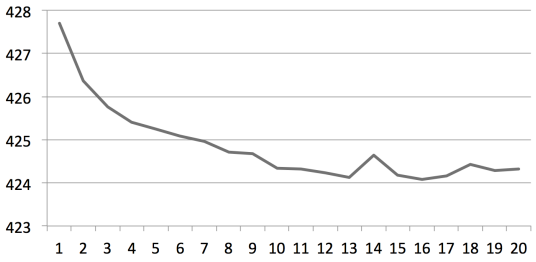

Figure 11: Average solution size as α varies from 1 to 20

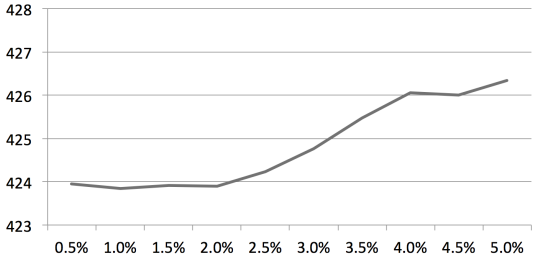

Figure 12: Average solution size as β varies from 0.5% to 5.0%

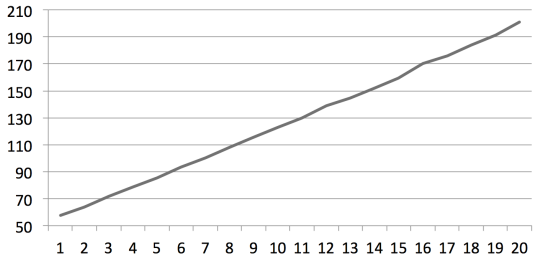

Figure 13: Average running time (milliseconds) as α varies from 1 to 20

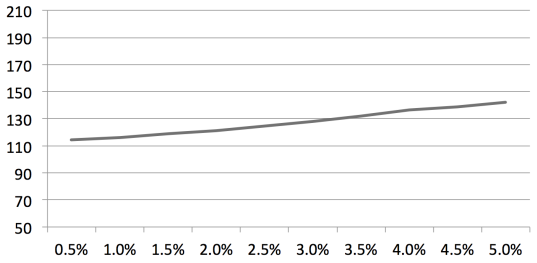

Figure 14: Average running time (milliseconds) as β varies from 0.5% to 5.0%

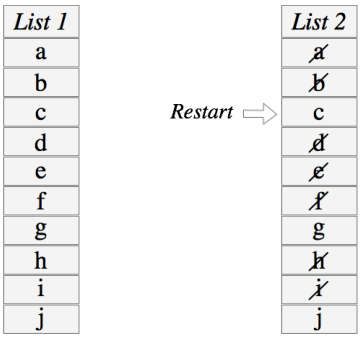

Figure 15: Illustration of Hybrid *CG* algorithm

Next, we applied the Wilcoxon signed rank test in order to test the null hypothesis that Hybrid *CG* and *ACO* have comparable solution quality. The results indicate that we cannot reject the null hypothesis. We suspect, however, that if we were to give Hybrid *CG* just a bit more running time, we would be able to conclude that, in a statistical sense, Hybrid *CG* outperforms *ACO*.

Finally, we point out that since Hybrid *CG* includes both intensification and diversification, it should be considered a new metaheuristic (see Blum and Roli [2]).

6 Conclusions and Future Work

In this paper, we started by studying the behavior of greedy algorithms and we identified several limitations. In response, we introduced carousel greedy – a new, generalized greedy algorithm. We were able to show that *CG* overcomes some of the major limitations of traditional greedy algorithms, at least on the problems studied. In addition, it is fast and widely applicable.

We test *CG* on several well-known problems in combinatorial optimization, including the minimum

Instance		Optimal	Carousel		Hybrid CG		Pilot	
n, ℓ	b	Size	Size	Time	Size	Time	Size	Time
50	4	13	**13.00**	4.95	**13.00**	21.10	13.35	31.55
50	5	10	11.05	4.20	**10.60**	20.90	10.90	28.80
50	10	5	6.10	3.40	5.85	20.25	**5.80**	19.65
50	15	4	4.30	3.05	**4.00**	19.65	**4.00**	15.90
50	20	3	3.90	2.80	**3.00**	19.05	**3.00**	13.40
75	4	19	19.75	13.85	**19.15**	62.00	20.40	152.40
75	7	11	11.90	11.89	**11.60**	68.75	12.00	117.35
75	15	5	6.30	9.35	**5.65**	50.80	**5.65**	82.70
75	25	3	4.30	7.15	**3.00**	45.60	**3.00**	57.40
100	4	25	**26.00**	31.15	**26.00**	158.80	27.55	472.25
100	10	10	11.55	23.40	**11.00**	147.35	11.85	310.75
100	20	5	6.60	18.05	**5.50**	104.25	5.85	219.50
100	30	4	4.90	15.45	**4.00**	93.60	**4.00**	168.25
150	4	38	38.85	108.10	**38.25**	666.70	42.20	2,513.35
150	15	10	12.05	62.75	**11.95**	371.75	12.05	1,269.40
150	30	5	6.95	46.10	6.15	277.00	**6.05**	901.85
150	45	4	5.05	40.20	**4.00**	241.45	**4.00**	700.80
200	4	50	51.50	260.20	**51.20**	1,588.20	56.20	7,963.50
200	20	10	12.95	131.25	**12.10**	744.35	13.00	3,474.40
200	40	5	7.10	98.55	**5.70**	579.00	5.80	2,515.55
200	60	4	5.10	92.15	**4.00**	549.80	**4.00**	2,439.45
# best solutions			**2**		**19**		**9**	

Table 12: Computational results for the $MLST$ problem using Hybrid CG

label spanning tree problem, the minimum vertex cover problem, the maximum independent set problem, and the minimum weight vertex cover problem. Although we used somewhat different values of the parameters α and β from one problem to the next, the values were chosen to ensure that the CG run times were no more than 5 to 10 times as large as the greedy run times. As we can see in Tables 1 and 2, there are different combinations of the parameters α and β for which we are able to obtain good results for the same problem, but the region of exploration is narrow (e.g., α between 2 and 8 and $\beta = 5\%$ or 10%). Nevertheless, to determine the best combination for a specific problem, some tuning is required. In future work, it might make sense to apply the automated tuning technique of Battiti and Brunato [1] to accomplish this goal.

In addition, we have developed a Hybrid CG algorithm which combines CG with the Pilot method in order to obtain a new metaheuristic. Preliminary computational tests indicate that Hybrid CG is faster and more effective than the standalone Pilot method.

Finally, in Appendix E, we show that CG is also applicable to statistics. In particular, we demonstrate how to use CG to improve variable subset selection using stepwise regression.

In future work, there are a number of interesting directions to explore. First, we expect to conduct more comprehensive computational studies of CG stepwise regression and Hybrid CG. Second, as pointed out by Glover et al. [20] and others, recent advances in technology have led to new applications involving very large to huge problem instances in which one–pass heuristics are required. Since CG is so efficient, we plan to test it on some very large problem (e.g., $MLST$) instances.

Nodes	Edges	Hybrid CG		ACO	
		Solution	Time	Solution	Time
50	50	1,290.1	18	**1,282.1**	63
	100	1,762.6	23	**1,741.1**	83
	250	**2,281.8**	24	2,287.4	97
	500	**2,669.6**	25	2,679.0	102
	750	2,962.9	20	**2,959.0**	125
	1000	**3,200.0**	24	3,211.2	117
100	100	2,576.6	79	**2,552.9**	273
	250	3,646.7	154	**3,626.4**	367
	500	**4,641.2**	161	4,692.1	433
	750	**5,070.3**	171	5,076.4	502
	1000	**5,513.7**	147	5,534.1	456
	2000	**6,062.9**	121	6,095.7	589
150	150	3,733.8	199	**3,684.9**	691
	250	4,827.8	312	**4,769.7**	891
	500	**6,200.7**	459	6,224.0	1,194
	750	**6,997.0**	457	7,014.7	1,042
	1000	**7,397.5**	416	7,441.8	1,206
	2000	**8,564.7**	389	8,631.2	1,103
	3000	**8,915.4**	390	8,950.2	966
200	250	5,668.8	581	**5,588.7**	1,674
	500	7,286.0	951	**7,259.2**	2,160
	750	8,352.9	1,052	**8,349.8**	2,602
	1000	**9,217.3**	1,104	9,262.2	2,221
	2000	**10,869.0**	929	10,916.5	2,437
	3000	**11,616.7**	858	11,689.1	2,497
250	250	6,281.5	882	**6,197.8**	2,273
	500	8,578.5	1,548	**8,538.8**	4,016
	750	9,873.1	1,950	**9,869.4**	4,047
	1000	**10,830.7**	2,095	10,866.6	3,755
	2000	**12,831.6**	1,867	12,917.7	3,942
	3000	**13,784.3**	1,750	13,882.5	4,276
	5000	**14,709.5**	1,693	14,801.8	3,842
300	300	7,468.8	1,534	**7,342.7**	4,322
	500	9,583.1	2,370	**9,517.4**	5,178
	750	**11,146.9**	3,022	11,166.9	6,055
	1000	**12,216.3**	3,425	12,241.7	6,231
	2000	**14,843.9**	3,391	14,894.9	6,488
	3000	**15,933.5**	3,191	16,054.1	6,299
	5000	**17,406.2**	2,838	17,545.4	6,558
# best solutions		**24**		**15**	

Table 13: Computational results for the $MWVC$ problem using Hybrid CG (times are in milliseconds)

Third, suppose we allowed CG to generate a handful of feasible solutions, rather than just one. If we revisit Figure 6, we see that for each of the five "iteration" rows, we have a partial solution with three labels. For each of these rows, we can identify the number of connected components when we add the edges with these labels to the empty graph on n nodes (remember, smaller is better). In addition to the CG solution, we might apply $MVCA$ to the k most promising partial solutions that CG encountered. The computational effort would be small, but the payoff may be nontrivial. We plan to experiment with this idea. Finally, we think there are numerous other problems in combinatorial optimization and statistics on which CG would work well. For example, it would be very natural to apply CG to the minimum feedback vertex set problem [5]. In future work, we might try to apply CG to a different class of optimization problems, like the traveling salesman problem [24], in which the cardinality of edges is fixed.

References

[1] R. Battiti and M. Brunato. *The LION Way. Machine Learning* plus *Intelligent Optimization*. Version 2.0, April 2015.

[2] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[3] S. Bouamama, C. Blum, and A. Boukerram. A population-based iterated greedy algorithm for the minimum weight vertex cover problem. *Applied Soft Computing*, 12(6):1632–1639, 2012.

[4] M. E. Captivo, J. C. Clímaco, and M. M. Pascoal. A mixed integer linear formulation for the minimum label spanning tree problem. *Computers & Operations Research*, 36(11):3082–3085, 2009.

[5] F. Carrabs, C. Cerrone, and R. Cerulli. A memetic algorithm for the weighted feedback vertex set problem. *Networks*, 64(4):339–356, 2014.

[6] C. Cerrone, R. Cerulli, and M. Gaudioso. Omega one multi ethnic genetic approach. *Optimization Letters*, 10(2):309–324, 2016.

[7] R. Cerulli, A. Fink, M. Gentili, and S. Voß. Metaheuristics comparison for the minimum labelling spanning tree problem. In *The Next Wave in Computing, Optimization, and Decision Technologies (B. Golden, S. Raghavan, and E.Wasil, eds.)*, pages 93–106. Springer, 2005.

[8] R.-S. Chang and L. Shing-Jiuan. The minimum labeling spanning trees. *Information Processing Letters*, 63(5):277–282, 1997.

[9] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[10] K. L. Clarkson. A modification of the greedy algorithm for vertex cover. *Information Processing Letters*, 16(1):23–25, 1983.

[11] S. Consoli, K. Darby-Dowman, N. Mladenović, and J. A. Moreno Pérez. Greedy randomized adaptive search and variable neighbourhood search for the minimum labelling spanning tree problem. *European Journal of Operational Research*, 196(2):440–449, 2009.

[12] S. Consoli, N. Mladenović, and J. M. Pérez. Solving the minimum labelling spanning tree problem by intelligent optimization. *Applied Soft Computing*, 28:440–452, 2015.

[13] S. Consoli and J. M. Pérez. Solving the minimum labelling spanning tree problem using hybrid local search. *Electronic Notes in Discrete Mathematics*, 39:75–82, 2012.

[14] N. R. Draper and H. Smith. Applied regression analysis 2nd ed. *New York New York John Wiley and Sons.*, 1981.

[15] C. Duin and S. Voß. The pilot method: A strategy for heuristic repetition with application to the Steiner problem in graphs. *Networks*, 34(3):181–191, 1999.

[16] T. A. Feo and M. G. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71, 1989.

[17] T. A. Feo and M. G. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995.

[18] M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937.

[19] F. Glover. Multi-start and strategic oscillation methods-principles to exploit adaptive memory. In *Computing Tools for Modeling, Optimization and Simulation (M. Laguna and J.L. González-Velarde, eds.)*, pages 1–23. Kluwer, 2000.

[20] F. Glover, B. Alidaee, C. Rego, and G. Kochenberger. One-pass heuristics for large-scale unconstrained binary quadratic problems. *European Journal of Operational Research*, 137(2):272–287, 2002.

[21] F. Harary. *Graph Theory*. Addison Wesley Publishing Company. Reading, MA, USA, 1972.

[22] J. D. Hirst, R. D. King, and M. J. Sternberg. Quantitative structure-activity relationships by neural networks and inductive logic programming. I. The inhibition of dihydrofolate reductase by pyrimidines. *Journal of Computer-Aided Molecular Design*, 8(4):405–420, 1994.

[23] D. S. Hochbaum. Efficient bounds for the stable set, vertex cover and set packing problems. *Discrete Applied Mathematics*, 6(3):243–254, 1983.

[24] K. L. Hoffman, M. Padberg, and G. Rinaldi. Traveling salesman problem. In *Encyclopedia of Operations Research and Management Science*, pages 1573–1578. Springer, 2013.

[25] D. S. Johnson and M. A. Trick. *Cliques, coloring, and satisfiability: second DIMACS implementation challenge, October 11-13, 1993*, volume 26. American Mathematical Soc., 1996.

[26] R. Jovanovic and M. Tuba. An ant colony optimization algorithm with improved pheromone correction strategy for the minimum weight vertex cover problem. *Applied Soft Computing*, 11(8):5360–5366, 2011.

[27] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations (R.E. Miller and J.W. Thatcher, eds.)*, pages 85–103. Plenum Press, New York, 1972.

[28] X. Ke. Bhoslib: Benchmarks with hidden optimum solutions for graph problems (maximum clique, maximum independent set, minimum vertex cover and vertex coloring) - hiding exact solutions in random graphs. *http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm*.

[29] S. O. Krumke and H.-C. Wirth. On the minimum label spanning tree problem. *Information Processing Letters*, 66(2):81–85, 1998.

[30] W. D. Mangold, L. Bean, and D. Adams. The impact of intercollegiate athletics on graduation rates among major NCAA division I universities: Implications for college persistence theory and practice. *The Journal of Higher Education*, 74(5):540–562, 2003.

[31] R. Ruiz and T. Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049, 2007.

[32] S. J. Shyu, P.-Y. Yin, and B. M. Lin. An ant colony optimization algorithm for the minimum weight vertex cover problem. *Annals of Operations Research*, 131(1-4):283–304, 2004.

[33] S. Voß and A. Fink. A hybridized tabu search approach for the minimum weight vertex cover problem. *Journal of Heuristics*, 18(6):869–876, 2012.

[34] S. Voßs, A. Fink, and C. Duin. Looking ahead with the pilot method. *Annals of Operations Research*, 136(1):285–302, 2005.

[35] Y. Xiong, B. Golden, and E. Wasil. A one-parameter genetic algorithm for the minimum labeling spanning tree problem. *IEEE Transactions on Evolutionary Computation*, 9(1):55–60, 2005.

[36] Y. Xiong, B. Golden, and E. Wasil. Improved heuristics for the minimum label spanning tree problem. *IEEE Transactions on Evolutionary Computation*, 10(6):700–703, 2006.

[37] T. Zhou, Z. Lü, Y. Wang, J. Ding, and B. Peng. Multi-start iterated tabu search for the minimum weight vertex cover problem. *Journal of Combinatorial Optimization*, to appear.

Appendix A

Description of $MVCA$

```
     Input: G(V, E, L). (V,E, and L are the sets of nodes, edges, and labels, respectively)
1    Let C ← ∅ be the set of used labels.
2    repeat
3        Let H be the subgraph of G restricted to V and edges with labels from C.
4        for all i ∈ L \ C
5            Determine the number of connected components when
             inserting all edges with label i in H
6        end for
7        Chose label i with the smallest resulting number of
         connected components and do: C ← C ∪ {i}.
8    until H is connected.
```

An $MVCA$ solution is given by any spanning tree built on the connected graph H.

Appendix B

Pseudo-code description of CG (minimization form)

```
     Input: I, G. (I is a generic input instance, G is a greedy algorithm)
1    Let S ← the solution produced by G
     (ordered by sequence of selection, the last selected elements are in the head)
2    R ← the partial solution produced by removing from head of S, β|S| elements
3    for α|S| iterations
4         remove from tail of R an element
5         using G, add an element to head of R
6    end for
7    using G, add elements to R to obtain a feasible solution
8    return R.
```

Pseudo-code description of CG (maximization form)

```
     Input: I, G. (I is a generic input instance, G is a greedy algorithm)
1    Let S ← the solution produced by G
     (ordered by sequence of selection, the last selected elements are in the head)
2    R ← the partial solution produced by removing from head of S, β|S| elements
3    for α|S| iterations
4         remove from tail of R an element
5         using G, add an element to head of R
6    end for
7    using G, add elements to R as long as the solution remains feasible
8    return R.
```

In rows 5 and 7, using G, means we select new elements to add to the partial solution R according to the greedy algorithm G.

Appendix C

Pseudo-code description of CG for minimum weight problems

```
     Input: I, G. (I is a generic input instance, G is a greedy algorithm)
1    Let S ← the solution produced by G
     (ordered by sequence of selection, the last selected elements are in the head)
2    R ← the partial solution produced by removing from head of S, β|S| elements
3    Let γ ← the objective function value associated with the partial solution R
4    for α|S| iterations
5         remove from tail of R one or more elements until
          the objective function value associated with the partial solution R is ≤ γ
6         using G, add an element to head of R
7    end for
8    using G, add elements to R to obtain a feasible solution
9    return R.
```

In this version of CG, we can remove more than one element from R (line 5). The reason is that this version of CG takes into account the weights of the elements.

Appendix D

Pseudo-code description of Iterated Greedy for the MLST problem

```
      Input: G. (G is the input graph)
1     Let S ← the solution produced by MVCA with input G
2     i ← 0
3     While i < 50
4           i ← i + 1
5           R ← the partial solution produced by removing 20% of the labels from S
6           Apply MVCA to the partial solution R to obtain a feasible solution S'
7           if |S'| < |S|
8                 S ← S'
9                 i ← 0
10    return S.
```

Appendix E

Stepwise Linear Regression

Multiple linear regression is a widely used statistical tool. The goal is to explain and predict outcomes, based on the available data. That is, can we use several independent or explanatory variables to predict a dependent variable? As an example, can we predict salinity in the Chesapeake Bay at any location, depth, and time of year?

When the number of independent variables in the data set is very large, it is extremely difficult, if not impossible, to determine the best subset of explanatory variables. A popular technique for selecting a good subset of variables is stepwise regression. Stepwise regression is an automated procedure for building a model; it successively adds or removes independent variables based upon F-tests or t-tests (see Draper and Smith [14] for details). There are three main approaches:

a) *Forward selection* begins with no variables. The procedure adds the most promising variable at each step, until additional improvements cannot be found.

b) *Backward elimination* starts with all of the candidate variables, deletes the variable such that the model is most improved, and repeats the process until the model can no longer be improved.

c) *Bidirectional stepwise regression* is a combination of the above two approaches. The procedure adds or deletes a variable at each step, until additional improvements cannot be found.

The first two approaches are greedy algorithms. Therefore, we introduce a *CG* stepwise regression algorithm in this section and compare it against the traditional statistical approach in a small preliminary experiment. We expect to design and execute a more comprehensive computational comparison in future work.

CG stepwise regression requires that the target number of explanatory variables (n) in the model be specified as input. A pseudo-code description is provided below.

```
    Input: I. (I is the set of explanatory variables)
    Input: n. (n is the number of explanatory variables you want in the model)
1   Let S ← model containing all the explanatory variables in I
2   R ← partial solution produced by removing from S, |S| − n elements
    according to the backward selection criteria
3   for αn iterations
4       remove from tail of R an explanatory variable
5       according to the forward selection criteria, add an element to head of R
6   end for
7   return R.
```

We used the software R for our experiments and we worked with two data sets. For each data set, we established two target values of n. The two basic sources were Mangold et al. [30] and Hirst et al. [22]. CG stepwise regression reduced the standard error or increased the R-squared value in each of the four cases, sometimes substantially. Although preliminary, these results are encouraging.

Appendix F

Pseudo-code description of Hybrid CG

```
    Input: L, k, l (respectively the set of elements, the maximum number of runs,
                    and the maximum number of levels (depth of exploration))
1   S ← ∅
2   P ← ∅
3   for i = 1 to l
4       S' ← ∅
5       P' ← ∅
6       k' ← k
7       L' ← L \ P
8       while(|L'| > 0 and k' > 0)
9           k' ← k' − 1
10          extract x from L'
11          compute the solution S'' using CG.
                CG start with the partial solution P ∪ x
                remove from L' all the elements added to the solution by CG
12          if S'' is better than S'
13              S' ← S''
14              P' ← P ∪ x
15      P ← P'
16      if S' is better than S
17          S ← S'
18  return S
```

Line 1: S represents the final solution.

Line 2: P is a partial solution.

Line 3: Start of the main loop (for l iterations). The only difference from one run to the next is the partial solution P.

Line 7: We remove from the list of candidate elements the elements in the partial solution creating L'.

Line 8: Start of the second loop which continues as long as there are elements in L' or our stop criteria has not been reached.

Line 10: Extract the element x from the list L'.

Line 11: We start CG with a partial solution equal $P \cup x$ (the previous selected elements plus the new

one). When CG selects a new element to build a solution, this element is removed from list L'.

Line 13: We store in S' the best solution identified in the second loop.

Line 14: We update the temporary partial solution P' by adding x.

Line 17: We store in S the best solution.

Appendix G

Mathematical Model for the Minimum Label Spanning Tree Problem

Let $G(N, E)$ be an undirected graph, where N is the set of nodes or vertices and E is the set of edges. Let L be a set of labels and assume each edge (i, j) is associated with a label $l_{ij} \in L$. For each $l \in L$, y_l is a Boolean variable that indicates if the label $l \in L$ is used in the solution. For each $(i, j) \in E$, x_{ij} is a Boolean variable that indicates whether or not edge (i, j) is used in the solution. The objective function (1) minimizes the number of labels in the solution. Constraints (2) are used to force $y_l = 1$ if an edge with label l is in the solution. Constraint (3) is used to ensure that there are $|N| - 1$ edges in any spanning tree. Constraints (4) are subtour elimination constraints.

$$(\textbf{MIP}) \qquad \min \sum_{l \in L} y_l \tag{1}$$

$$\sum_{(i,j) \in E \mid l_{ij} = l} x_{ij} \leq (|N| - 1) y_l \qquad \forall l \in L \tag{2}$$

$$\sum_{(i,j) \in E} x_{ij} = (|N| - 1) \tag{3}$$

$$\sum_{(i,j) \in E(S)} x_{ij} \leq |S| - 1 \qquad \forall S \subseteq N, S \neq \emptyset \tag{4}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i, j) \in E \tag{5}$$

$$y_l \in \{0, 1\} \qquad \forall l \in L \tag{6}$$

Mathematical Model for the Minimum Vertex Cover Problem

Let $G(N, E)$ be an undirected graph, where N is the set of nodes or vertices and E is the set of edges. For each $i \in N$, x_i is a Boolean variable that indicates if the vertex i is in the solution. The objective function (7) minimizes the number of vertices in the solution. Constraints (8) ensure that there is at least one endpoint per edge in the solution.

$$(\textbf{MIP}) \qquad \min \sum_{i \in N} x_i \tag{7}$$

$$x_i + x_j \geq 1 \qquad \forall (i, j) \in E \tag{8}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i, j) \in E \tag{9}$$

Mathematical Model for the Maximum Independent Set Problem

Let $G(N, E)$ be an undirected graph, where N is the set of nodes or vertices and E is the set of edges. For each $i \in N$, x_i is a Boolean variable that indicates if vertex i is in the solution. The objective function (10) maximizes the number of vertices in the solution. Constraints (11) ensure that at most one vertex per edge will be selected.

$$(\textbf{MIP}) \qquad \max \sum_{i \in N} x_i \tag{10}$$

$$x_i + x_j \leq 1 \qquad \forall (i,j) \in E \tag{11}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in E \tag{12}$$

Mathematical Model for the Minimum Weight Vertex Cover Problem

Let $G(N, E)$ be an undirected graph, where N is the set of nodes or vertices and E is the set of edges. In addition, we associate a weight $w(i)$ to each vertex i. For each $i \in N$, x_i is a Boolean variable that indicates if the vertex i is in the solution. The objective function (13) minimizes the sum of the weights of the vertices in the solution. Constraints (14) ensure that there is at least one endpoint per edge in the solution.

$$(\textbf{MIP}) \qquad \min \sum_{i \in N} w(i) x_i \tag{13}$$

$$x_i + x_j \geq 1 \qquad \forall (i,j) \in E \tag{14}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in E \tag{15}$$