2020

# Improvements to Iterated Local Search for Microaggregation

Tracy Bierman
*Nova Southeastern University*, tracy.bierman@gmail.com

# Improvements to Iterated Local Search for Microaggregation

by

Tracy Alan Bierman

A dissertation to be submitted in partial fulfillment of requirements
for the degree of Doctor of Philosophy
in
Computer Science

College of Computing and Engineering
Nova Southeastern University

2020

We hereby certify that this dissertation, submitted by Tracy Bierman conforms to acceptable standards and is fully adequate in scope and quality to fulfill the dissertation requirements for the degree of Doctor of Philosophy.

| | |
|---|---|
| Michael J. Laszlo, Ph.D<br>Chairperson of Dissertation Committee | June 22, 2020<br>Date |
| Francisco J. Mitropoulos, Ph.D.<br>Dissertation Committee Member | June 22, 2020<br>Date |
| Sumitra Mukherjee, Ph.D.<br>Dissertation Committee Member | June 22, 2020<br>Date |

Approved:

| | |
|---|---|
| Meline Kevorkian, Ed.D.<br>Dean, College of Computing and Engineering | June 22, 2020<br>Date |

College of Computing and Engineering
Nova Southeastern University

2020

An Abstract of a Dissertation Submitted to Nova Southeastern University in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

# Improvements to Iterated Local Search for Microaggregation

by
Tracy Alan Bierman
May 2020

Microaggregation is a disclosure control method that uses *k*-anonymity to protect confidentiality in microdata while seeking minimal information loss. The problem is NP-hard. Iterated local search for microaggregation (ILSM) is an effective metaheuristic algorithm that consistently identifies better quality solutions than extant microaggregation methods. The present work presents improvements to local search, the perturbation operations and acceptance criterion within ILSM.

The first, ILSMC, targets changed clusters within local search (LS) to avoid vast numbers of comparison tests, significantly reducing execution times. Second, a new probability distribution yields a better perturbation operator for most cases, significantly reducing the number of iterations needed to find similar quality solutions. A third improves the acceptance criterion by replacing the static balance between intensification and diversification with a dynamic balance. This helps ILSM escape local optima more quickly for some datasets and values of *k*.

Experimental results with benchmark data show that ILSMC consistently reduces execution times significantly. Targeting changed clusters within LS avoids vast numbers of unproductive tests while allowing search to concentrate on more productive ones. Execution times are decreased by more than an order of magnitude for most benchmark test cases. In the worst case it decreased execution times by 75%. Advantageously, the biggest improvements were with the largest datasets. Perturbing clusters with higher information loss tend to reduce information loss more. Biasing the perturbation operations toward clusters with higher information loss increases the rate of improvement by more than 50 percent in the earliest iterations for two of the benchmarks. Occasionally accepting worse solutions provides diversification; however, increasing the probability of accepting worse solutions closer in quality to the current best solution aids in escaping local optima. This increases the rate of improvement by up to 30 percent in the earliest iterations. Combining the new perturbation operation with the new acceptance criterion can further increase the rate of improvement by as much as 20 percent for some test cases. All three improvements are orthogonal and can be combined for additive effect.

## Acknowledgements

I dedicate this work to my father, Larry. From my earliest days he instilled in me a love of education. He was not afforded many opportunities, so he made sure I was. Through the years, no one followed my progress more closely, asking for an update nearly every time we met. Thank you, dad, I love you.

I am grateful to my advisor Dr. Michael Laszlo, who has shown considerable patience. His advice has proven a necessary ingredient in the success of this dissertation. Thank you to my committee members Dr. Frank Mitropoulos and Dr. Sumitra Mukherjee. Dr. Mitropoulos introduced me to the many crosscutting aspects of programming. Dr. Mukherjee built upon my love of algorithms and feature vectors whose objective functions I will attempt to optimize for many years to come. Dr. Laszlo helped me visualize a world of data in compendious and expressive ways. My three favorite professors, my three favorite classes.

I am also in awe of the incredible and clever little algorithm Dr. Laszlo and Dr. Mukherjee created and allowed me the privilege to make better. They almost left no room for improvement. I am most grateful for the “almost” part.

My wife Kaley gets the credit for starting me on this journey. It feels like yesterday when she surprised me with a date to a graduate school open house and made me apply. I remember presenting myself as indifferent, but secretly I was overjoyed. Thank you, my love, for encouraging me to take the leap, for being my resting place along the way and single handedly caring for two rambunctious children for nights on end.

To my son Joseph and my daughter Katherine I greatly appreciate your boundless joy, inventive minds and unconditional love. You are the reason for almost everything I do. My advice is not to wait as long as I did to achieve your dreams, but also know that good things do eventually come to those who persevere.

# Table of Contents

# List of Tables

**Tables**

# List of Figures

**Figures**

# Chapter 1

# Introduction

## Background

This research studied the problem of securing the release of data from statistical databases against disclosure of confidential information. Specifically, it studied techniques that improve both speed and quality of secured information.

Microdata consists of data records containing personally sensitive and private information of individuals and/or organizations (Mateo-Sanz & Domingo-Ferrer, 1998). Enormous amounts of microdata are widely collected. Researchers and others with legitimate purposes seek access to the latent information within such data. However, privacy of the data subjects is of utmost concern. Ethics, privacy laws and possible compensatory and punitive damages due to inappropriate disclosure are all considerations when disclosing data. Most disclosure methods that maximize privacy protections rely on a key principle: change the released data such that individual identities can no longer be deduced from the data. If one can deduce an identity, then they might be able to infer something confidential using associated or linked microdata (Adam & Worthmann, 1989). Additionally, a desirable characteristic of a disclosure method is the minimal loss of legitimately usable information within the data (Adam & Worthmann, 1989). The characteristics of minimizing both inferences and loss of legitimate information result in conversely competing design motivations; it is simple to minimize inferences if one ignores loss of information, but significantly more difficult to simultaneously minimize both with speed and in terms of quality.

Microaggregation is a disclosure method that seeks to maximize privacy protection while also minimizing loss of legitimate information. It is a statistical disclosure control technique that relies on data modification to provide *k*-anonymity (Samarati, 2001; Sweeney, 2002) to the individual subjects within the data. *K*-anonymity provides a guarantee that an individual's information cannot be distinguished from *k* minus one other individuals (Sweeney, 2002). It is achieved by partitioning the set of records into groups with a minimum of *k* and a maximum of 2*k*-1 records (for fixed integer *k*). This is called the *k*-partition. It then replaces the records in each group by the group's mean value. Information loss is measured by the sum of the squared Euclidean distances between the value of the original records and their associated group's mean value. Finding good solutions to the microaggregation problem, ones with acceptable information loss, is known to be NP-hard (Oganian & Domingo-Ferrer, 2001).

Practical application of microaggregation usually involves large numbers of records (Chang, Li, & Huang, 2007). This leads to large numbers of groups within the partition because *k* tends to be relatively small. It is an optimization problem with the goal of searching and finding a partition with overall minimum information loss. Heuristic search algorithms are known to find good solutions (Chang et al., 2007; Panagiotakis & Tziritas, 2011). In addition, meta-heuristic search (Blum & Roli, 2003) and specifically iterated local search is known to provide significant improvements (Laszlo & Mukherjee, 2015).

Specifically, the microaggregation problem is a combinatorial optimization problem defined by points in Euclidian space, partitions, constraints and an objective function to be optimized. All possible feasible combinations of variable assignments that satisfy the constraints are the candidates that make up the global search space. The subset of those candidates with optimal objective function value make up the set of solutions. In this report the term *solution* generally refers to the local optimum solution. The most optimal solution from this set would be the globally optimal solution. However, the microaggregation problem is NP-hard and no polynomial time algorithms exist; thus, finding the globally optimal solution may need exponential computation time to find. Therefore, practical methods for solving the microaggregation problem do not guarantee the globally optimal solution but settle for good enough solutions in exchange for significantly lower execution run time (Laszlo & Mukherjee, 2015).

Local search is described as a heuristic method for finding better and possibly good enough solutions for many computationally hard combinatorial optimization problems (Blum & Roli, 2003). Local search, as applied to the microaggregation problem, starts with some initial candidate (i.e. a *k*-partition within a subset of the global search space.) The search space is defined by a neighborhood structure and a set of rules governing moves from the current selected solution to a neighbor solution. A search of neighbors is performed based on the defined set of move rules. Generally, a move is made to a neighbor when it is found to be better than the current best solution. Alternatively, a group of neighbors is searched and then a move is made to the best one better than the current best solution. Searching continues until no better neighbors can be found. This is a form of iterative improvement where the current best solution is progressively improved through a successive number of moves to better neighboring solutions.

Often in practice, search spaces are so large that heuristic search methods only search a very small subset of the possible candidates (Laszlo & Mukherjee, 2015). Therefore, while the final solution will be a local optimum, it will likely not be known if it is the global optimum solution. It is also quite possible that all the candidates searched are poor candidates resulting in an inferior local optimum.

One way to increase the probability of finding a good solution is to perform many searches over the global search space (Laszlo & Mukherjee, 2015). The starting point of these local searches is important. If these small search spaces represented by the starting candidates are not diverse, the searches can become entrapped by the same local optimum solution. Key to increasing the probability of finding a good solution is to perform many searches starting from a diverse number of candidates within the global space (Blum & Roli, 2003). Making starting candidates sufficiently diverse minimizes the overlap of associated sub search spaces. In practice, sufficient diversity can nearly guarantee escape of the previous local optimum solution. Maximizing diversity provides the highest probability of finding a different local optimum. Nonetheless, the latest local optimum solution may be a worse solution not a better one. Selecting candidates at random is a way to maximize diversity. The best solution from among the resulting group of local optima is then selected. Performing repetitive searches with random starting candidates is known as local search within a random restart regime (Laszlo & Mukherjee, 2015).

Further improvement can be achieved by combining two or more basic heuristic methods into higher-level frameworks (Blum & Roli, 2003). These frameworks are commonly referred to as metaheuristics and their intent is to increase search efficiency and effectiveness over singular heuristic methods. Iterated local search (ILS) is a specific metaheuristic method that employs local search with two additional higher-level heuristics (Blum & Roli, 2003). Local search is run on a random *k*-partition to obtain an initial local optimum solution. The solution is saved in memory as the current best solution. Then a loop is entered. A perturbance heuristic is then used to perturb the solution and change it a little. The resulting *k*-partition is then used as input to another run of local search. Another local optimum solution is found and returned. Then a decision is made whether to accept or not accept this new solution. If it is better than the current best solution it is automatically accepted, and it replaces the current best solution in memory. If it is not better a second heuristic is used to decide whether to accept or reject the inferior solution. It then returns to the beginning of the loop. If the solution was accepted it is perturbed and the loop proceeds again as described above. However, if the solution was rejected, then the current best solution is perturbed instead, and the loop proceeds again as describe above. The Iterations continue until some stop criterion is met.

The perturbance heuristic should be formulated on the observation that better solutions have many attributes and characteristics in common with the better and best solutions from previous iterations(Laszlo & Mukherjee, 2015). The advantage is each successive iteration takes advantage of information from previous iterations. In simple, each new iteration of local search is started with a solution similar to the best solutions found so far.

This leads to the other high-level heuristic for accepting which solution is used for the next iteration. The acceptance heuristic (known as the acceptance criterion) could be as simple as only accepting a new solution if it is better than the current best solution (Blum & Roli, 2003). However, if only the best solutions are accepted then only the current best solution is perturbed. Then the danger arises of being entrapped by the current best solution. Therefore, the acceptance heuristic should add some diversity to expand the search space. It does this by accepting some solutions found by LS which are not necessarily the best solution found so far. Accepting a solution not as good as the current best solution is often referred to as a worsening move. If after one or more of worsening moves, and a new best solution is not found, the heuristic could determine to reject the new solution. This effectively undoes or returns the search back to the current best solution.

From the discussions above two counter motivations are evident. Local search in a random restart regime provides maximum diversity by starting every iteration of LS with a random partition. It does not seek to exploit previous search experience. This maximizes the probability local maxima are escaped. Conversely, the perturbation heuristic in ILS seeks to exploit previous search experience by producing new solutions "very similar" to the best solutions already found. Nonetheless, if too little change results from the perturbance the search will likely lead to the same local optimum as before. Key to the ILS metaheuristic is the degree to which the solution and the perturbed result are similar yet dissimilar (Blum & Roli, 2003).

These countervailing forces are referred to as diversification and intensification. "Diversification generally refers to the exploration of the search space" and "intensification refers to the exploitation of the accumulated search experience" (Blum & Roli, 2003). Both will be described in greater detail in the Literature Review section. Just note they are important concepts in metaheuristics. They are complementary yet likely contrarian and counter motivating and must be balanced. They generally determine the behavior of the metaheuristic (Blum & Roli, 2003). A metaheuristic is the smart balance of these two concepts. They guide and direct the underlying subordinate search heuristic. The intent is to improve performance over just the use of the subordinate heuristic alone. A metaheuristic can statically balance diversification versus intensification, or it can dynamically change the balance. It can also use a combination of the two.

The researchers Laszlo and Mukherjee (2015) use both diversification and intensification to great effect in their iterated local search for microaggregation (ILSM). It consistently identifies better quality solutions than other extant microaggregation methods (Laszlo & Mukherjee, 2015). Core to their approach is a novel local search heuristic (LS). LS starts with any valid partition, and it monotonically produces a valid solution with equal or less information loss. Also, LS does not change the partition size, the number of groups within the partition. It should be noted that LS is not likely to find the globally optimal solution and may even produce inferior solutions. To help avoid inferior solutions, the researchers use LS within the context of their iterated local search metaheuristic ILSM. Key to their approach are their perturbation operations which change only a small portion of the solution yet guarantee escape of local optima by changing the size of the partition.

**Problem Statement**

Previous research on ILSM by Laszlo and Mukherjee (2015) describe a problem where “at present, all pairs of clusters are tested, yet relatively few are likely to interact.” Local Search (LS) processes a fixed set of cluster pairs and tests for the beneficial swapping and shifting of points between the cluster pairs. These tests are the costliest part of local search. Nonetheless, few tests are likely to result in swaps or shifts. Most tests can be avoided if the cluster pairs that have a possible interaction can be efficiently identified and used to inform the next iteration within LS.

Previous research on ILS, detailed in the Literature Review section, suggests dynamically adjusting balances between intensification and diversification is generally more effective than simple static balances. ILSM, while an effective ILS microaggregation algorithm, uses simple fixed balances between intensification and diversification in two key areas that lower information loss. The perturbation operations use a simple fixed uniform distribution to select clusters to involve in perturbations. The acceptance criterion accepts solutions from LS using a simple fixed uniform distribution.

As ILSM runs, it has been observed that information loss becomes unevenly distributed. Many of the groups within the *k*-partition result in relatively low information loss while many others remain with relatively high information loss. Also, it is observed that perturbations that involve clusters with higher information loss tend to result in larger reductions of information loss. This suggests that the perturbances should be biased toward clusters with higher information loss; however, ILSM selects clusters to perturb with a uniform probability distribution. If probability distributions biased toward selecting clusters with higher information loss can be efficiently constructed, the perturbation operations can be guided toward more promising clusters and larger corrections.

Accepted solutions within ILS can be the current best solution or some similar solution with a small amount of additional information loss. An inverse correlation has been observed between the size of the additional information loss and the likelihood the accepted solution leads to a new best solution. This suggests that ILS should bias accepting solutions from LS toward solutions with lower additional information loss. If probability distributions biased toward smaller additional information loss can be efficiently constructed, the acceptance function can be guided toward accepting solutions more likely to escape local optima and lead to a new best solution.

Three novel improvements are defined in this study. They address the three problems discussed above. Most unnecessary tests within LS are avoided, significantly reducing execution times compared to LS. A new perturbation operation and acceptance criterion are more effective in most test cases, reducing the number of iterations needed to reach similar solutions compared to ILSM.

## Dissertation Goal

The goal of this research was to develop improvements to ILSM capable of equal or better quality microaggregation partitions while using significantly less execution time. Benchmarks were used to demonstrate that these improvements significantly improve performance of ILSM.

Most unnecessary shift and swap tests are now avoided within LS significantly reducing execution times. ILSM statically balances intensification and diversification in two key areas. Improvements presented here dynamically balance intensification and diversification in these two areas. Selecting clusters to perturb with a biased probability distribution significantly improves effectiveness of the perturbation operation. A second dynamic probability distribution improves the effectiveness of accepting solutions. All three improvements are complementary and additive, demonstrating even greater improvement when used together.

## Research Questions

The following questions posed here are answered in Chapter 5. The answers are supported by the experimental results using benchmark datasets where the experiments and methodology are outlined in Chapter 3.

RQ1: Concerning the efficiency of LS, how does tracking changed clusters help avoid testing cluster pairs compared to LS where all cluster pairs are tested?

RQ2: Concerning the effectiveness of the perturbation operations, how does the use of a dynamically biased probability distribution for selecting clusters to perturb compare to a static uniform distribution?

RQ3: Concerning the effectiveness of the acceptance criteria, how does the effectiveness of a dynamically biased probability distribution for accepting solutions within ILSM compare to a static uniform distribution?

## Relevance and Significance

The information latent in statistical databases is of immense value to social science and statistical database researchers (Fienberg, 2005; United Nations General Assembly, 2014). It is often referred to as social science data in the literature (Fienberg, 2005). Providing researchers and analysts access to this kind data it is seen as logical and beneficial to society. Researchers should try to release as much as possible without undue disclosure risks (Fienberg, 2005). Individuals and organizations are often the original source of this information. This leads to confidentiality and privacy concerns of individuals and organizations associated with data. Addressing these concerns is widely seen as necessary before dissemination can occur. The United Nations felt it was so important it passed a general assembly resolution regarding the principles surrounding the release of its own statistical information (United Nations General Assembly, 2014).

> United Nations General Assembly Resolution
>
> Fundamental Principles of Official Statistics
>
> Principle 1. Official statistics provide an indispensable element in the information system of a democratic society, serving the Government, the economy and the public with data about the economic, demographic, social and environmental situation. To this end, official statistics that meet the test of practical utility are to be compiled and made available on an impartial basis by official statistical agencies to honour citizens' entitlement to public information.
>
> Principle 6. Individual data collected by statistical agencies for statistical compilation, whether they refer to natural or legal persons, are to be strictly confidential and used exclusively for statistical purposes.

Feinberg (2005) defines Confidentiality – "Broadly, a quality or condition accorded to statistical information as an obligation not to transmit that information to an unauthorized party." Confidentiality is rooted in privacy, where privacy is defined as follows – “The right of individuals to control the dissemination of information about themselves (Fienberg, 2005).” The concern with releasing statistical information is described as “The attribution of information to a data provider, whether it be an individual or organization (Fienberg, 2005)” that is confidential. Identity and attribute disclosure are the two general types of disclosure (Fienberg, 2005). Identity disclosure happens when an individual or organization can be identified by analysis of released data and/or possibly enable by matching it to other known data. Attribute disclosure happens when analysis of released data can result in a higher likelihood that an attribute can be inferred about an individual or organization. Identity disclosure often facilitates attribute disclosure, so both are usually considered together. Disclosure is spoken of in terms of likelihood or probability of discovery. Any meaningful releases of data would increase the likelihood or risk of inferences through analysis of the data. This statistical nature of the problem is why the literature refers to disclosure in terms of confidentiality, disclosure limitation and statistical disclosure control instead of absolute protections and preventions. See Fienberg (2005) for an in depth examination of confidentiality and disclosure limitation.

The confidentiality problem is innate in all information that has privacy concerns and it is commensurate with the mix of privacy concern, nature of the release and legitimate use (Adam & Worthmann, 1989; Fienberg, 2005). Releasing census data, medical information, sales and commerce information, and social media data all pose innate confidentiality and privacy concerns (Adam & Worthmann, 1989; Campan & Truta, 2009; Sweeney, 2002). Data stewards of statistical databases could even have legal obligations that must be upheld. Some legal obligations come with the possibility of penalties if information is disclosed ("Health Insurance Portability and Accountability Act of 1996," 1996). The Health Insurance Portability and Accountability Act (HIPAA), passed by Congress in 1996, mandates standards and guidelines for the protection and confidential handling of health and medical information. It also establishes penalties for inappropriate disclosure and mishandling of information.

Medical advancements, specifically in the personalization of medicine, like molecular medicine, systems biology and genomics are improving healthcare and raising the importance of medical information (Adam & Worthmann, 1989; Sweeney, 2002). Medicine is becoming more effective, safer while becoming even more personal and tailored. Improvements and the increasing ubiquity of information technology in the overall practice of medicine along with its embedding in medical equipment have resulted in mass collection of personal medical information. Combined with analytical advances it is now practical for researchers to perform large-scale biomedical data mining (Adam & Worthmann, 1989; Sweeney, 2002).

Data stewards have a vital responsibility to maintain individual confidentiality when releasing statistical data (Duncan, Elliot, & Salazar-González, 2011). The United Nations affirms its importance by specifically addressing this issue within Principle 6 of its Fundamental Principles of Official Statistics: "Individual data collected by statistical agencies for statistical compilation, whether they refer to natural or legal persons, are to be strictly confidential and used exclusively for statistical purposes" (Duncan et al., 2011; United Nations General Assembly, 2014). The National Institute of Standards and Technology have a published guide to protecting the confidentiality of personally identifiable information (National Institute of Standards and Technology, 2014). The

Federal Information Security Management Act (FISMA) decrees all Federal government agencies must follow the NIST guidelines.

Legal obligations like those imposed by laws like HIPPA and ethical responsibilities imposed by the medical, legal and other professions greatly impeded the release of data for legitimate use. Also, the possibility of financial liability associated with a disclosure gives financial disincentives for owners and stewards of data to release it. If confidentiality protections against disclosure risks could be proved and data utility retained with computationally efficiency the barriers to releasing data would be greatly reduced. The amount and variety of released data would be greatly increased. Without a doubt, these increases would result in commensurate increases in information discoveries and in a significantly greater overall social benefit.

**Barriers and Issues**

Best-known solutions to the microaggregation problem structure it as a combinatorial optimization (CO) problem. Methods and algorithms applied to CO problems are generally classified as complete or approximate. Complete methods find globally optimal solutions in bounded time for finite instances of the problem. The microaggregation problem is NP-hard and no polynomial time algorithms exist; thus, complete methods are likely in the worst case to need exponential computation time to find the global optimum. Approximate methods seek to significantly reduce execution time but do so by trading the guarantee of finding the global optimal solution for finding solutions considered good enough. Approximate methods tend to use either constructive or local search approaches. Constructive methods typically start with an empty solution and incrementally build it out to a complete solution. They are usually much faster than local search methods but usually result in lower quality in comparison.

Therefore, practical methods for solving the microaggregation problem do not guarantee the globally optimal solution but settle for good enough solutions in exchange for significantly lower execution time. Current solutions can typically only search a small portion of the overall solution space. Search heuristics must be efficiently employed to affectively explore these extremely large spaces.

The perturbation operations for ILSM use a uniform distribution to select clusters to perturb. This uniform distribution is quite simple and easily calculated with a simple call to the random function. This research found that introducing bias into the perturbation operations improved the effectiveness of individual ILSM iterations, but construction of biased probability distributions in real-time was costly. Approximating the biased probability distribution with sampling proved just as effective with little computational costs in comparison.

The acceptance criterion for ILSM uses a uniform distribution to select solutions to accept from LS. As in the perturbation operations, this uniform distribution is also quite simple and easily calculated with a simple call to a random function. This research found that introducing bias into the acceptance criteria improved the effectiveness of individual ILSM iterations but constructing biased probability distributions from history was costly and not possible early in the run of the algorithm. However, using an exponential probability distribution as a function of the additional information loss proved effective with some of the benchmarks. It also had little computational costs in comparison to constructing probability distributions in real-time.

# Chapter 2

# Review of the Literature

## Microaggregation and *k*-Anonymity

The literature describes microdata as sets of data records associated with data subjects, including both individuals and organizations (Mateo-Sanz & Domingo-Ferrer, 1998). Microdata are widely collected and their number is expected to grow exponentially as computer and networking technology advances (Sweeney, 2002). Statistical analyses on microdata have long led to new and significant discoveries of information and to the social good (Adam & Worthmann, 1989). This has led governments to increasingly encourage release of it to the public. It has also led to ever increasing demand from researchers (Fienberg, 2005).

When microdata are used raw, the discoveries and conclusions of most analyses are easily linked to the associated data subjects (Samarati, 2001; Sweeney, 2002). It is for this reason most microdata are considered sensitive and have confidentiality and privacy concerns. An obvious step in protecting a data subject's anonymity is removing or obfuscating explicit identifiers like names, telephone numbers, addresses, and social security numbers.

Obviously, if the explicit identifiers were not removed, disclosure happens by definition. However, what is not obvious is that microdata can still be somewhat easily exploited, and disclosures made even after explicit identifiers are removed. Implicit identifiers within the microdata, called quasi-identifiers (Samarati, 2001; Sweeney, 2002), can be matched within generally available public information. Matches within this publicly available information generally leads to discovery of explicit identifiers. Identity disclosure occurs as a result. This in turn facilitates further disclosure when matched up with released microdata specifically resulting in attribute disclosure. Much publicly available information is nefariously useful and easily obtained from governments (Sweeney, 2002). Examples include census data and voter rolls.

A fundamental principle of disclosure control methods is that disclosure risks are guaranteed to be below some acceptable level (Fienberg, 2005). The concept is to sufficiently obscure or mask the data such that disclosure risks are reduced (Adam & Worthmann, 1989; Sweeney, 2002). Classically, the disclosure risks are assessed after the obscurations are made. To do this, attempts are made to match the quasi-identifiers to other publicly available data. A second fundamental but competing principle is to retain as much as possible of the utility and usefulness of the original data. This can be conceptualized as strategically making obscurations with the most efficient changes possible. The focus is on minimizing the loss of information.

There is a tension between these two opposing or conflicting objectives (Domingo-Ferrer & Torra, 2005). The conflict is between minimizing the disclosure risk by discarding information and maximizing the information by retaining information. In other words, achieving enough obscurations while minimizing obscurations. These two principles are by their nature inherently statistical (Fienberg, 2005). *K*-anonymity is well suited at solving this tension (Domingo-Ferrer & Torra, 2005).

The importance of releasing statistical and social science along with the innate requirements of protecting confidentiality has led to a broad assortment of disclosure control methods over many years. Early methods stressed protection over quality. Much of the early research used simple masking of the data through generalizations and suppression (Adam & Worthmann, 1989; Samarati, 2001; Sweeney, 2002). The technique required the masked data to be tested by matching it with other publicly available data and analyzing the results for disclosure. The process was iterated, and additional masking performed till the information was deemed sufficiently protected.

Around 2001, the literature starting making a strong case that $k$-anonymity, for a given $k$, provided a kind of guarantee against disclosure risk (Samarati, 2001; Sweeney, 2002). It still achieved anonymity through generalizations and suppression; however, testing the masked data by matching it to other publicly available information was conveniently no longer required (Domingo-Ferrer & Torra, 2005). The $k$-anonymity method guarantees a statistical level of disclosure risk. The value of $k$ became the representative measure of data protection against disclosure risk. Later literature begins to show a consensus forming around $k$-anonymity as a superior method (Samarati, 2001; Sweeney, 2002); one of the reason given is that $k$-anonymity "neatly" reduces the "tension" between the objectives of data protection and data utility. It allows focus to be placed on the mission of minimizing information loss since efforts need only to singularly satisfy the $k$-anonymity constraint. Much of the research now concentrates on improving the quality of $k$-anonymity methods and their computational efficiency. (Chang et al., 2007; Domingo-Ferrer, Sebe, & Solanas, 2008; Domingo-Ferrer & Torra, 2005; Hansen & Mukherjee, 2003; Kokolakis & Fouskakis, 2009; Laszlo & Mukherjee, 2015; Panagiotakis & Tziritas, 2013).

Microaggregation is a class of methods simply defined as grouping microdata into groups of *k* individuals where similar individuals are placed in the same group (Mateo-Sanz & Domingo-Ferrer, 1998). Placement is done by criterion that optimizes a measure of similarity and homogeneity within groups. The value of *k* is typically a set value; however, groups can contain more than *k* individuals. No individual's attribute variables should dominate a group. To mitigate an individual from dominating a group, individuals may be added making the group larger than *k*, until the individual no longer dominates (Mateo-Sanz & Domingo-Ferrer, 1998). Also, groups that contain 2*k* or more individuals can always be split without increasing information loss. After the groups are established a representative aggregate (an average individual) is derived for each group. For each group, the original variables in each record are replaced with the variable values from the representative (Mateo-Sanz & Domingo-Ferrer, 1998). Consensus is also forming that microaggregation like *k*-anonymity offers compelling benefits. It turns out microaggregation is well suited to satisfy *k*-anonymity (Domingo-Ferrer & Torra, 2005). The constraint of *k*-anonymity is easily met by setting the minimum group size to the value of *k*.

Constructing a microaggregation is a partition problem (Mateo-Sanz & Domingo-Ferrer, 1998; Oganian & Domingo-ferrer, 2001); however, it differs from hierarchical and *k*-means clustering. Typical clustering constructs a partition with a fixed number of groups while the sizes of groups are not constrained. Microaggregation constructs a partition of groups where the number of groups is not constrained while the sizes of the groups have a minimum constraint of size $k$. Both univariate cases of clustering (Brucker, 1978) and microaggregation (Hansen & Mukherjee, 2003) have polynomial-time algorithms while the multivariate cases for both are known NP-hard (Oganian & Domingo-ferrer, 2001).

Microaggregations can be categorized as either fixed size with fixed size groups and variable size with group sizes greater than or equal to $k$ (Mateo-Sanz & Domingo-Ferrer, 1998). Most methods create fixed sizes. However, methods that produce variable group sizes can reduce informational loss but usually at additional computational cost. Some methods use heuristic search and tend to have better results. Recent work has applied a meta-heuristic approach called iterated local search. A number of heuristic and meta-heuristic search methods generate good partitions for larger $k$-partitions typically seen in practical applications (Chang et al., 2007; Domingo-Ferrer, Martinez-Balleste, Mateo-Sanz, & Sebe, 2006; Domingo-Ferrer & Mateo-Sanz, 2002; Goldberger & Tassa, 2010; Hansen & Mukherjee, 2003; Kokolakis & Fouskakis, 2009; Laszlo & Mukherjee, 2007, 2015; Oommen & Fayyoumi, 2010; Panagiotakis & Tziritas, 2013; Rebollo-Monedero, Forné, & Soriano, 2011).

Fixed sized methods generally start with a given number $k$ and a pool of unselected points of size $X$; then strategically selecting $k$ neighboring points from the pool they place them in a new group; iterating this they create new groups till there are $\lfloor X/k \rfloor$ groups. Then they strategically distribute the remaining unselected points, which will be less than $k$, throughout the existing groups. What distinguish these fixed size methods are the different strategies used to form groups.

Heuristic methods typically leverage searchable structures to discover local optima. For example some model paths in a network where the path corresponds to the construction of an optimal partition (Hansen & Mukherjee, 2003). Heuristics are then devised for making good decisions selecting between branches while traversing the network structure. The second is to construct a neighborhood of similar solutions and search the neighborhood for the local optima using a heuristic to score each neighbor (Blum & Roli, 2003). Neither approach provides globally optimal solutions and on occasion can produce bad solutions. Using an iterative restart regime can sometimes help since only the best solution from one of the iterations is used (Blum & Roli, 2003).

This brief review of the literature leads to the following conclusions. Consensus has formed favoring $k$-anonymity since it can provide data protection as a statistically measurable level of disclosure risk. The simple value of $k$ becomes the representative measure of disclosure risk where any combination of quasi-identifies will always return at least $k$ identical individuals. In addition, it does not require disclosure risk assessment, as does the classical masking approach. $K$-anonymity is seen as a novel and elegant way to reduce the tension between the conflicting objectives of data protection and data utility (Domingo-Ferrer & Torra, 2005). This reduces the challenge by achieving an independent quantitative standard for data protection; thereby, allowing efforts to be concentrated on data utility.

The literature also shows that microaggregation can easily satisfy the $k$-anonymity constraint. It is simply a partition with clusters of at least size $k$ or greater to some maximum size. Yet the structure is conducive to reducing information loss. It neatly aids in the decoupling of data protection from the mission of reducing information loss. For this it is considered novel and elegant. An additional benefit is that microaggregation frameworks provide a more natural and efficient fit in achieving *k*-anonymity; it does so across the widest variety of attribute types when compared to classical generalization and suppression (Domingo-Ferrer & Torra, 2005).

Microaggregation also narrows the challenge of reducing information loss to a challenge of minimizing a well-defined objective function. For most methods, the objective function and quantitative measure is the sum of squares error criterion. The sum of squares error, made available by microaggregation, has become the independent quantitative standard for information loss like *k*-anonymity has for data protection.

Currently ILSM consistently identifies solutions with lower information loss than other known microaggregation algorithms. It employs a metaheuristic explorative search algorithm described as Iterated Local Search for Microaggregation. Iterated local search (ILS) is a general technique described as both a simple and powerful metaheuristic. ILS applies a local search heuristic to an initial candidate to find an initial local optimum solution. A second heuristic perturbs or strategically changes that solution. Using the perturbed result as input, it performs local search again resulting in another unique local optimum solution. At this point a third heuristic chooses (also referred to as the "acceptance criterion") one of the previous solutions to perturb. Then the cycle of accepting, perturbing and restarting local search is repeated until some termination criteria are met (e.g. a set number of overall iterations or set number of iterations since last improvement.)

Blum and Roli (2003) describe good iterated local search metaheuristics as having the following characteristics. The local search heuristic should be effective. Constructing a good or good enough initial starting candidate should be fast. The main purpose of the perturbation is to define the amount of change to the new local optimum solution and where those changes are made. This is described as strength in the literature where more strength roughly correlates to more intensification. Also, the perturbation must sufficiently guarantee local search escapes the new local optimum and finds a unique local optimum solution. The perturbations should exploit the natural tendencies within the microdata where the best solutions tend to be near good solutions. Intensifying search near the current best solution should find a better solution quicker than restarting with just another random partition. Strength can be fixed or vary. Strength may vary with the size of the problem or be used to adjust the balance of intensification versus diversification as needed. The acceptance heuristic based on the new local optimum should use diversification to counterbalance the intensification of the perturbations. It can be described as between the two extremes, always accepting the new solution and accepting the new solution only if it is an improvement.

ILSM achieves these objectives well. Its local search (LS) when run in a random restart regime produces better quality solutions on benchmark datasets than most extant heuristics. It constructs random initial candidates extremely fast although it does not concern itself with quality. The random candidates are considered good enough. The perturbations change the size of the microaggregation partition guaranteeing that local optima are escaped. By removing a group and dispersing the members amongst the remaining groups or making a new group from excess points leaves the candidate relatively unchanged from the latest solution. The result is a mostly similar (near) partition to the original. The probability of accepting new solutions over the best-found solution is set at eighty percent. This was the best value found to biases the search toward diversification as a counterbalance to the intensification of the perturbations.

## Summary

This literature suggests several areas for possible improvement. Laszlo and Mukherjee (2015) state that “the most costly part of LS is testing whether a pair of clusters can swap or shift points.” One of the improvements in this report uses an approach to significantly reduce the number of cluster pairs that must be tested. ILSM uses a static approach to strength within the perturbations. Blum and Roli (2003) state “that variable strength is in general more effective.” A second improvement uses sampling to increase strength in the perturbation operations. It was found to significantly reduce the number of iterations for equivalent results. ILSM uses a fixed approach in its acceptance criterion, if the new solution is not an improvement it accepts it eighty percent of the time. Blum and Roli (2003) suggest an adaptive acceptance criterion which exploits search history can be more effective than fixed approaches. A third improvement accepts new solutions using delta information loss to dynamically balance intensification versus diversification. It too was found to reduce the number of iterations for equivalent results.

# Chapter 3

# Methodology

This chapter discusses the methodology for evaluating novel improvements to local search (LS), the perturbation operations and the acceptance criterion within ILSM. The improvements are described and explained in this section in context of the prior research.

## Overview

The first improvement efficiently identifies cluster pairs that do not interact allowing vast numbers of costlier tests to be avoided. The second uses a new probability distribution within the perturbation operations to select better perturbations. The new probability distribution biases the selection of clusters toward clusters demonstrated to result in a higher reduction of information loss. The third replaces within the acceptance criterion, the static probability of accepting a solution with one that is dynamically varied. Dynamically varying the probability of acceptance adjusts the balance between intensification and diversification and was demonstrated to be beneficial.

Algorithms and experiments reproducing ILSM and LS were recreated as presented in Laszlo and Mukherjee's (2015) previous work. The same set of experiments were performed for implementations of LSC, ILSMC, ILSM with *bDissolve* and ILSM with *dAcceptanceCriterion*. Results were recorded for quality (percentage of information loss) with every iteration of ILSM & ILSMC, and the execution elapsed times to complete the runs.

The experiments were performed using the same three benchmark datasets used by Laszlo and Mukherjee (2015). These widely used datasets were used to evaluate many existing microaggregation heuristics. As in Laszlo and Mukherjee's study (2015), the data used is normalized so all attributes have the same proportionate effect on group formation. Each attribute has a mean of zero and a standard deviation of one. The three benchmarks are list in Table 1.

| **Name** | **Number of Points (n)** | **Dimensions of the points** | **Description** |
|---|---|---|---|
| *Tarragona* | 834 | 13 | Comprising figures of 834 companies in the Tarragona area of Spain. Data corresponds to the year 1995. Examples of variable attributes: fixed assets, current assets, uncommitted funds, paid-up capital, short-term debt and sales. |
| *Census* | 1080 | 13 | Obtained on July 27, 2000 using the Data Extraction System of the U. S. Bureau of the Census. Examples of variable attributes: adjusted gross income, employer contribution for health insurance and federal income tax liability. |
| *EIA* | 4092 | 10 | Obtained from the U.S. Energy Information Authority. Data corresponds to the year 1996. Examples of variable attributes: sales to residential consumers, sales to commercial consumers, sales to industrial consumers and sales to all consumers. |

**Table 1:** Benchmark Datasets

The values of $k$ ($k$ = 3,4,5,6 and 10) for generating the microaggregation problem instances were the same values used in previous work. These are values typically used in practical microaggregation problems.

**Prior Research and Improvements**

As stated before, microaggregation provides $k$-anonymity to individuals in a dataset by replacing each group of records and associated attributes with a single mean record. The mean record consists of attributes where each attribute is the mean of the group's related attributes. It does this for all groups of individuals in the partition. The downside of this substitution is the loss of information. Simply stated, microaggregation is the problem of constructing a partition that provides $k$-anonymity yet minimizes information loss. Several assumptions and definitions need to be discussed to provide specific structure and more concise description of the prior research. First the microaggregation problem discussed here is limited to datasets with numerical attributes. A dataset with records of $d$ numerical attributes is modeled as a set of points, $X$ in $\mathbb{R}^d$, where the $d$-tuple of real number attributes is modeled as a point vector in $d$-dimensional Euclidean space. A $k$-partition $P^k(X)$, later denoted by $P$, is defined as a partition of $X$, where every group $C_i$ in the partition contains at least $k$ points from $X$ and every point is included in one and only one group. Using inputs $X$ and $k$, sum of squared errors (SSE) can now be defined as $\mathrm{SSE}(P) = \left(\sum_{C_i \in P^k(X)} \sum_{x \in C_i} \Delta(x, \bar{C}_i)\right)$, where $C_i$ is a group in the partition and $\Delta(x, \bar{C}_i)$ is the squared Euclidean distance from $x$ to its group mean $\bar{C}_i$. Given $X$ and $k$ the microaggregation problem can now be succinctly described as constructing a $k$-partition $P$ that minimizes SSE. To be consistent with Laszlo and Mukherjee (2105), the measure of quality is the standardized information loss for a $k$-partition $P^k(X)$ where $X$ is the set of points and standard percentage information loss is defined as $\mathcal{H}\left(P^k(X)\right) = \frac{\mathrm{SSE}\left(P^k(X)\right)}{\sum_{x \in X} \Delta(x, \bar{X})} \times 100$. Values for $\mathcal{H}\left(P^k(X)\right)$ range from zero to one hundred percent..

**Local Search (LS)**

Local Search (LS) is a heuristic local search method for the microaggregation problem presented by Laszlo and Mukherjee (2015). LS was designed as a non-increasing (monotonic) search heuristic that iteratively searches a neighborhood of $k$-partitions. During its search LS converges to a local optimum while the neighborhood structure enforces partition feasibility, which means every neighbor in the neighborhood is a valid $k$-partition.

While LS can be used standalone, LS was designed to be complementary to the ILSM (Laszlo & Mukherjee, 2015). The neighborhood structure utilized by LS is also utilized by the ILSM perturbation operations. ILSM complements LS, by expanding the search space that LS searches. It does this by perturbing the solutions from LS and increasing or decreasing the partition size. Changing partition size is highly effective in keeping the overall algorithm from cycling.

Essential to operation of LS is the definition of its neighborhood structure $\mathcal{N}$ (Laszlo & Mukherjee, 2015). First, for a given $X$ and $k$, $\mathcal{P}^k(X)$, denoted by $\mathcal{P}$, is define as the set of all $k$-partitions. Then $2^{\mathcal{P}^k(X)}$, denoted by $2^{\mathcal{P}}$, is defined as the power set of $\mathcal{P}$, the set of all possible subsets of $\mathcal{P}$. The neighborhood structure for LS is realized by the function $\mathcal{N}\colon\ \mathcal{P} \to 2^{\mathcal{P}}$. This function maps every $k$-partition $P^k(X)$, denoted by $P$, where $P \in \mathcal{P}$ to one of the subsets in the power set $2^{\mathcal{P}}$. In short, $\mathcal{N}$ maps every $k$-partition to its neighborhood. Neighborhoods are subsets of $k$-partitions from the set $\mathcal{P}$. A partition $P' \in \mathcal{P}$ is a neighbor of $P$ (i.e. $P' \in \mathcal{N}(P)$), if $P'$ meets the following criteria: $P'$ can be obtained from $P$ by performing at most a single application of either of the following two operations: (a) A swap, the operation of transposing a pair of points from two groups of $P$

; (b) A shift, the operation of moving a point from some group $C \in P$ where $|C| > k$ to another group $C' \in P$ where $C' \neq C$. Note: The partition $P$ is always a member of its own neighborhood.

LS operates in the following way (Laszlo & Mukherjee, 2015). LS starts with an initial $k$-partition $P$. LS then calls an *update* procedure passing $P$ as a parameter. The *update* procedure then returns a $k$-partition with a lower SSE or a copy of the original $P$. The return value is assigned to $P'$. When $P'$ and $P$ are equivalent, LS exits. In this way LS starts from any $k$-partition $P$ and successively generates $k$-partitions with monotonically decreasing information loss. It does through a sequence of local improvement moves within *update*. The value returned from LS is called a local optimum solution. The pseudocode for LS follows:

```
LS(P) {
   while(true)
      P' ← update(P);
      if (SSE(P') = SSE(P)) return P';
      P ← P';
}
```

Given a $k$-partition $P$, consider the set $\{\{C_i, C_j\} \mid C_i \in P, C_j \in P, C_i \neq C_j\}$, the set of every group pair where $C_i \neq C_j$. The *update* procedure can be described as a traversal that visits every pair in the set just once in random order. With each visit there are many applications of the two operations to the points within the groups. The two operations are the *swap* and *shift* operations mentioned above. The *update* procedure operates in the following way (Laszlo & Mukherjee, 2015). A $k$-partition $P$ is passed in as a parameter. It then generates the set of all possible group pairing $\{C_i, C_j\}$ where $C_i \in P$, $C_j \in P$ and $C_i \neq C_j$. The *update* procedure then generates a random ordering from the set of group pairings. It then starts with the first group pair in the ordering and generates the set of all possible point pairs $\{x, y\}$ where $x \in C_i$ and $y \in C_j$. It then applies the swap operation for the first point pair forming the neighbor $P'$ where $P' \in \mathcal{N}(P)$. If $\text{SSE}(P') < \text{SSE}(P)$ the move is committed by replacing $P$ with $P'$. The *update* procedure continues performing a swap for every point pair $\{x, y\}$ and committing moves where there are improvements. It then applies the shift operation for the first point $x$, where $x \in C_i$, shifting it to $C_j$. The neighbor $P'$ is formed, where $P' \in \mathcal{N}(P)$. If $\text{SSE}(P') < \text{SSE}(P)$ then the move is committed by replacing $P$ with $P'$. It continues performing a shift and test for every point in $C_i$ and committing moves where there are improvements. The same is performed for $C_j$. The *update* procedure continues processing all the group pairs in order until they are exhausted. It is possible after application of all the operations that $P'$ never has a lower SSE than the initial *P*; therefore, the returned value from *update* is either an improved *k*-partition with lower SSE or a copy of the input parameter unchanged. In this way the *update* procedure starts from any valid *k*-partition $P$ and successively generates *k*-partitions with monotonically decreasing information loss (Laszlo & Mukherjee, 2015). LS repeatedly calls *update* passing in the improved *k*-partition from the previous call. Once *update* does not improve the *k*-partition all subsequent calls to *update* will also fail to improve the *k*-partition. At this point there is no reason to continue and LS stops and exits. The value returned from LS is called a local optimum solution. It is a *k*-partition where the groups have been optimized into tightly bound clusters. In the following discussions the term cluster will be interchangeable with group.

The $k$-partitions resulting from a swap and shift operation are denoted respectively by $swap(P, C_i, C_j, x, y)$ and $shift(P, C_i, C_j, x)$ and defined as follows (Laszlo & Mukherjee, 2015):

$$swap(P, C_i, C_j, x, y) = \{C_q | C_q \in P, q \neq i, j\} \cup \{C_i \backslash \{x\} \cup \{y\}\} \cup \{C_j \backslash \{y\} \cup \{x\}\}$$

$$shift(P, C_i, C_j, x) = \{C_q | C_q \in P, q \neq i, j\} \cup \{C_i \backslash \{x\}\} \cup \{C_j \cup \{x\}\}$$

Everything necessary is now defined to illustrate the pseudocode for $update$:

```
update(P) {
   for every pair of cluster {Ci , Cj} where Ci ∈ P, Cj ∈ P and Ci ≠ Cj
        for every pair of points x ∈ Ci and y ∈ Cj
           if (SSE(swap(P, Ci, Cj, x, y)) < SSE(P))
              P ← swap(P, Ci, Cj, x, y);
        for every point x ∈ Ci
           if (SSE(shift(P, Ci, Cj, x)) < SSE(P))
              P ← shift(P, Ci, Cj, x);
        for every point y ∈ Cj
           if (SSE(shift(P, Cj, Ci, y)) < SSE(P))
              P ← shift(P, Cj, Ci, y);
   return P;
}
```

Laszlo and Mukherjee (2015) identified two opportunities within *update* to improve efficiency. The first is the optimization of $\mathrm{SSE}\big(swap(P, Ci, Cj, x, y)\big) < \mathrm{SSE}(P)$ within a new function *swapTest* and *shiftTest*. The *swapTest* and *shiftTest* check if a swap or shift would be beneficial, resulting in lower SSE. They decide the following:

$$swapTest(P, C_i, C_j, x, y) = \mathrm{SSE}\left(swap(P, C_i, C_j, x, y)\right) < \mathrm{SSE}(P)$$

$$shiftTest(P, C_i, C_j, x) = \mathrm{SSE}\left(shift(P, C_i, C_j, x)\right) < \mathrm{SSE}(P)$$

To help explain *swapTest*, consider the simple Boolean method that would just perform the swap and then decide $\mathrm{SSE}(P) > \mathrm{SSE}\left(swap(P, C_i, C_j, x, y)\right)$. If the answer is true replace $P$ with the new $k$-partition. Similar statements also hold for *shiftTest*. These are naïve implementations of *swapTest* and *shiftTest*; however, the cost of *swap, shift*, and SSE operations justify well-designed tests. Laszlo and Mukherjee (2015) developed two efficient tests that avoid the costly calculations of SSE (see theorems below).

The second opportunity is the quick identification of cluster pairs that do not interact. If a cluster pair can be identified efficiently, $O(k^3)$ time for swaps and $O(k^2)$ time for shifts can be avoided for each cluster pair identified. The new function *maySwap* decides if any pair of points from the pairs of clusters could possibly satisfied *swapTest* (Laszlo & Mukherjee, 2015). In short, it is a quick reject test. If *maySwap* is not satisfied, then no pair of points from the cluster pairs could possibly satisfy *swapTest* and it can be avoided altogether. Each time *maySwap* is not satisfied many calls to *swapTest* are avoided. Similar statements hold for *mayShift* and *shiftTest*. The *maySwap* and *mayShift* quick reject tests are based on the understanding that two clusters can be too far apart to favorably interact. So, if a pair of clusters are sufficiently far apart, the associated calls to *swapTest* and *shiftTest* will fail and can be avoided. A naive implementation for *maySwap* and *mayShift* would be to always assume they are close enough to interact and always return true; however, a well-designed quick reject test is justified by the cost of testing cluster pairs for interaction (Laszlo & Mukherjee, 2015).

Four theorems have been developed that show these fundamental Boolean-valued functions (*swapTest*, *shiftTest*, *maySwap* and *mayShift*) are easily implemented as efficient tests. Experimentation has shown them to greatly improve computational efficiency. The four theorems follow, see Laszlo and Mukherjee (2015) for proofs.

$$swaptest\left(C_i, C_j, x, y\right) = \Delta(x, \overline{C}_i) + \Delta(y, \overline{C}_j) + \left(\frac{1}{|C_i|} + \frac{1}{|C_j|}\right)\Delta(x, y) > \Delta(x, \overline{C}_j) + \Delta(y, \overline{C}_i)$$

$$shifttest(C_i, C_j, x) \; = \frac{|C_i|}{|C_i|-1}\,\Delta(x, \bar{C}_i) > \frac{|C_j|}{|C_j|+1}\,\Delta(x, \bar{C}_j)$$

$$mayswap(C_i, C_j) \; = \; (r_i + r_j) > \delta(\bar{C}_i, \bar{C}_j)$$

$$mayshift(C_i, C_j) \; = \frac{|C_i|}{|C_i|-1}\,r_i^2 > \frac{|C_j|}{|C_j|+1}\,\left(\delta(\bar{C}_i\,\bar{C}_j) - r_i\right)^2$$

The fundamental Boolean tests, *swapTest*, *shiftTest*, *maySwap*, and *mayShift* can now be shown in *update*. The pseudocode follows:

```
update(P) {
   for every pair of cluster {Ci , Cj} where Ci ∈ P, Cj ∈ P and Ci ≠ Cj
      if (maySwap(Ci, Cj))                  // quick reject test
         for every pair of points x ∈ Ci and y ∈ Cj
            if (swaptest(P, Ci, Cj, x, y))    // swap test
               P ← swap(P, Ci, Cj, x, y);
      if (mayShift(Ci, Cj))                 // quick reject test
         for every point x ∈ Ci
            if (|Ci| > k)
               if (shifttest(P, Ci, Cj, x))     // shift test
                  P ← shift(P, Ci, Cj, x);
      if (mayShift(Cj, Ci))                 // quick reject test
         for every point y ∈ Cj
            if (|Cj| > k)
               if (shifttest(P, Cj, Ci, y))     // shift test
                  P ← shift(P, Cj, Ci, y);
   return P;
}
```

## Avoiding Costly Tests with LSC and *updateC*

Even though there are efficient implementations for *swapTest* and *shiftTest*, and for the quick reject tests, *maySwap* and *mayShift*, the costliest part of LS is still testing whether a pair of clusters can swap or shift points. The *update* procedure is basically a loop that visits every possible pair of clusters in random order and tests them for beneficial swaps and shifts. The number of cluster pairs visited is $\frac{|P|(|P|-1)}{2}$ where $P$ is the *k*-partition passed as a parameter to *update*. However, Laszlo and Mukherjee (2015) state that few cluster pairs are likely to interact. They conclude that if it were possible to efficiently identify all the pairs of clusters that do interact, many more tests could be avoided. Remember the *maySwap* and *mayShift* identify pairs of clusters that do not interact, a subtle but crucial difference. During this study it was generally observed that few clusters interacted after a relatively small number of calls to *update* confirming their earlier understanding.

Consider new procedures LSC and *updateC* that introduces the variable *targetClusters* into the LS and *update* procedures. This new variable is a set of targeted clusters used to seed the generation of cluster pairs within the *updateC* procedure. The original *update* targets all the clusters in the *k*-partition every time it is called. Passing the complete set of clusters from the *k*-partition into *updateC* makes it equivalent to *update*. The LSC and *updateC* call signatures are also changed to accept this variable as a parameter. The *updateC* procedure is also changed to return a tuple which includes the improved *k*-partition and this *targetClusters* variable. The pseudocode for LSC and *updateC* follows.

```
LSC(P, targetClusters) {
  while(true)
    (P', targetClusters) ← updateC(P, targetClusters);
    if (SSE(P') = SSE(P)) return P';
    P ← P';
}
```

```
updateC(P, targetClusters) {
  changed ← Ø;
  for every cluster pair {Ci, Cj} where Ci ∈ targetClusters, Cj ∈ P and Ci ≠ Cj
    if (maySwap(Ci, Cj))                    // quick reject test
      for every pair of points x ∈ Ci and y ∈ Cj
        if (swaptest(P, Ci, Cj, x, y))     // swap test
          P ← swap(P, Ci, Cj, x, y);
          changed ← changed ∪ {Ci, Cj};
    if (mayShift(Ci, Cj))                   // quick reject test
      for every point x ∈ Ci
        if (|Ci| > k)
          if (shifttest(P, Ci, Cj, x))      // shift test
            P ← shift(P, Ci, Cj, x);
            changed ← changed ∪ {Ci, Cj};
    if (mayShift(Cj, Ci))                   // quick reject test
      for every point y ∈ Cj
        if (|Cj| > k)
          if (shifttest(P, Cj, Ci, y))      // shift test
            P ← shift(P, Cj, Ci, y);
            changed ← changed ∪ {Ci, Cj};
  return (P, changed);                      // changed clusters are targeted
}
```

The contents of the *targetClusters* can range from the complete set of clusters in the $k$-partition down to a set containing just a couple of clusters. A set of cluster pairs is then constructed within *updateC* by pairing every cluster in *targetClusters* with every cluster in the $k$-partition. Again, when the *targetClusters* variable contains the complete set of clusters from the $k$-partition, the set of cluster pairs generated within *updateC* is equivalent to the set cluster pairs generated within *update*. The advantage is gained when *targetClusters* only contains a couple of clusters and the number of cluster pairs subsequently generated is significantly smaller.

The target clusters are simply the changed clusters from the previous call to *updateC*. It suffices for now to say LSC should be called with the entire set of clusters in the $k$-partition (i.e. a call would look like `LSC(P, {C | C∈P})`). Later it will be shown how passing in a set of targeted clusters to LSC is beneficial. The *updateC* procedure changes clusters whenever a swap or shift occurs. It also follows that both clusters involved in either operation are changed; therefore, *targetClusters* will never have a single cluster. They are then both added to the *changed* variable if they do not already exist. Upon return from the call, *updateC* returns a tuple, the new *k*-partition *P* and the set of clusters changed during the call. The changed clusters become the target clusters. LSC then calls *updateC* again and passes as input parameters the two outputs from the previous call.

It was stated above that few clusters interact after a relatively small number of calls to *update*. It is not uncommon for the number of changed clusters to drop significantly within the first 25% of calls to *updateC*. The number of cluster pairs tested in *updateC* is a function of the size of the *targetClusters* set from the previous call. The set of cluster pairs consists of the possible combinations of the changed clusters with all the clusters in the partition. The size of this set is calculated by the formula $|c||P| - \frac{|c|(|c|+1)}{2}$ where *c* is the set *targetClusters*. For small values of $|c|$, which is likely most of the time, the size approaches $|c||P|$.

Each call to *update* must test $\frac{|P|(|P|-1)}{2}$ cluster pairs. For comparison take a partition of 1000 clusters, the original LS and *update* would test nearly 500K pairs for every call to *update*. If most of the calls to *updateC* have only 4 target clusters, then the approximate number of cluster pairs tested is 4K compared to the nearly 500K pairs tested for every call to *update*. Thus, in this example many of the calls to *updateC* result in most of the tests (i.e. *maySwap*, *mayShift*, *swapTest* and *shiftTest*) being avoided.

This approach for avoiding tests is valid in part because the four fundamental tests *maySwap*, *swapTest*, *mayShift* and *shiftTest* are functions (when given the same input they always return the same output). Because a pair of clusters is visited only once during a single call to *update*, once a pair is visited, it is not visited again until the next call to *update*. There are two possible outcomes that result from a visit to a pair of clusters. The first is when either a swap or shift occurs. The outcome is that both clusters are changed. The second is when neither a swap nor shift occurs during the visit. This second outcome is more interesting. If two or more clusters remain unchanged by the current call to *update*, in the next call to *update*, when an unchanged cluster is paired with another unchanged cluster, the pair can be skipped. This is because the four fundamental tests when given the same input always return with the same output. If the previous call, and the current call to *update* up to this point, have not resulted in a change to either cluster, then the pair is identical to when they were tested in the previous call. Since the pair failed the tests then (i.e. *maySwap*, *swapTest*, *mayShift* and *shiftTest*) they will do the same now.

Recall that the *update* procedure can be described as a traversal that visits in random order every cluster pair in the set, visiting each pair just once. During each visit, the pair is tested, and beneficial swaps and shifts are performed. Now, consider only the possible traverses where the traversal first visits all the cluster pairs with two unchanged clusters. Then follow that with all the visits to the pairs with one or two changed clusters. We know that all the visits to the first group of pairs with unchanged clusters will result in no swaps or shifts. Only in the second group where there is at least one changed cluster per pair is there a possibility for swaps and/or shifts. If we limit *updateC* to just traversing the second group, then only cluster pairs with one or more changed clusters in the previous call need to be visited and tested. All the clusters pairs that had both clusters return unchanged in the previous call to *updateC* can be skipped and have their tests avoided in the current call to *updateC*. While this is a loosening of the random order requirement, there were no discernable negative effects associated with limiting the random traverses to only those orderings that meet the above requirement.

The first research question for this study asks with respect to the efficiency of LS, can the clusters with which a given cluster potentially interacts be efficiently identified, thereby avoiding a vast number of pairwise tests? Using the above approach many pairwise tests can indeed be avoided. For example, if *updateC* is called with only 2 changed clusters then the number of cluster pairs tested is $(2|P| - 3)$ versus $\frac{|P|(|P|-1)}{2}$ for *update*. For a partition with 1000 clusters that is approximately 2K cluster pairs for *updateC* compared to 500K for *update*. When few clusters interact, this improvement avoids vast numbers of tests. However, its real strength is when it is coupled with ILSMC, discussed below.

**Iterated Local Search for Microaggregation (ILSM)**

Local Search (LS) starts with any *k*-partition and searches the space around it to find local optima. It does so by identifying basins of attraction with good local optima (Blum & Roli, 2003). The effectiveness and performance of a local search heuristic depends on the starting point, the size of problem search space and the relative size of basins of attraction to good local optima. LS shows sensitivity to its starting point. Section 1 noted that LS, while currently the best of all known heuristics on benchmark problems, is not likely to find a globally optimal solution and may even produce inferior local optima. Blum and Roli (2003) state that running local search in a random restart regime can help overcome this weakness but may be less effective when the size of search space increases and/or relative size of basins of attraction decrease past a certain point. LS when run in a random restart regime demonstrates it can consistently find good solutions and sufficiently overcome any sensitivity to starting points. However, to find even better performance Laszlo and Mukherjee employed their LS within the iterated local search (ILS) metaheuristic resulting in ILSM.

ILSM is like running LS in a random restart regime. However, the ILS metaheuristic approach does not restart local search with a random starting point. Instead starting points are chosen from along a trajectory. This is known to extend effectiveness compared to the random restart regime. Trajectory is modeled by using local optima from past searches. The trajectory guides the search, providing necessary direction to ever improving basins of attraction. It would be ideal, if it were possible, to model trajectory as a basin of attraction itself. A special neighborhood might be constructed where all the neighbors are themselves local optima (Blum & Roli, 2003). Then one would just search this neighborhood to find the global optimum. Unfortunately, no viable neighborhood structures of just local optima are known (Blum & Roli, 2003).

Nonetheless, a meta-structure representing trajectory of local optima can be envisioned. Consider an operation bounded on either side by two requirements (Blum & Roli, 2003). The first would be to sufficiently perturb the local optima to achieve enough difference. The second, which would be in opposition, would be to not perturb so much as to make it indistinguishable from a random starting point. The concept behind trajectory is to provide enough change to escape entrapment by the local optimum but preserve as much of the momentum that defines the trajectory. In this way the perturbation operation becomes a balance of not too much perturbation causing the trajectory to be lost and not too little so that local search undoes the perturbation.

ILSM perturbation operations are based on the concepts of the meta-structure discussed above. ILSM starts with a uniform random *k*-partition. ILSM applies LS to the initial candidate to find an initial local optimum solution. A perturbation operation then perturbs the solution. Using the perturbed result as input, LS is restarted resulting in a new unique local optimum solution. At this point the ILSM acceptance criterion is applied. If the new solution is also the new best-found solution it is accepted and saved as the current best solution. Otherwise, the new solution is inferior and accepted with an eighty percent probability. For the other twenty percent, the new solution is rejected, and the current best solution is returned to. The cycle of perturbing, restarting LS and accepting is repeated. The overall process is repeated until some termination condition (e.g. 5000 iterations) is met and the best solution is returned. The pseudocode for ILSM follows:

```
ILSM(P) {
    P ← LS(P);
    bestP ← P;
    while (not terminationCondition)
        P' ← perturb(P);
        P'' ← LS(P');
        P ← acceptanceCriterion(P'', bestP);
        If (SSE(P) < SSE(bestP))
            bestP ← P;
    return bestP;
}
```

Laszlo and Mukherjee (2015) designed their perturbation operations *dissolve* and *distill* to be complementary to their local search method LS. While LS preserves the number of clusters in the resulting partition, the perturbation operations either decrease or increase the number of clusters in the perturbed partition while leaving most of the partition unchanged. Changing the number of clusters thus prevents entrapment by the previous local optimum. This satisfies the first stated objective of perturbation operations, to sufficiently perturb solutions to assist in escape of local optima. The pseudocode for the *perturb* procedure follows.

```
perturb(P) {
    if (size(P) = minP) return distill(P);
    else if (size(P) = maxP) return dissolve(P);
    else if (random < 0.5) return distill(P);
    else return dissolve(P);
}
```

The *minP* and *maxP* values represent the smallest and largest possible sizes for *k*-partition within ILMS. This keeps cluster sizes in a range from $k$ points to $2k - 1$ points, thus satisfying the constraint *k*. It follows that the acceptable range of *k*-partition sizes ranges from $\left\lceil \frac{n}{2k-1} \right\rceil$ to $\left\lfloor \frac{n}{k} \right\rfloor$. Nonetheless, *minP* is usually set higher because better solutions tend to come from the larger partitions. A value for *minP* that restricts partition sizes to a top percentile (e.g. top quintile) is practical (Laszlo & Mukherjee, 2015).

The *dissolve* operation removes one cluster at random from the *k*-partition and strategically distributes the points contained within the cluster throughout the partition. It follows that the size of some cluster $C$ may be greater than *k* and contain $|C| - k$ extra points. Excess points are those extra points in groups larger than *k* and farthest from the mean center not counting the *k* closest points. The *distill* operation constructs a new cluster for the *k*-partition from the excess points if there are enough. This operation can only be performed if the size of the partition is less than the maximum size of $\left\lfloor {}^{n}/_{k} \right\rfloor$ where $n$ is the number of points in the dataset. It chooses one of the excess points at random to seed the new cluster. Then the $k-1$ best excess points are moved to the new cluster with the centroid recalculated between every move. After either *dissolve* or *distill* most of the $k$-partition is left unchanged. This satisfies the second objective of perturbation operations, to preserve as much of what makes the solution good. Overall, the perturbation operations utilize intensification, they exploit the accumulated search experience. The balance between intensification and diversification in both *dissolve* and *distill* is statically set by the uniform probability distributions used within the operations.

The *dissolve* operation maintains feasibility (results in a valid $k$-partition) as it decreases the number of clusters in the partition. As illustrated in the pseudocode below, *dissolve* removes a random cluster $C$ and distributes the associated points $p \in C$ to nearby clusters that result in the lowest cost. Each point $p$ is placed within the cluster with the closest centroid represented by the symbol $\overline{D}$. The distance function $\delta(p, \overline{D})$ returns the Euclidean distance between the point $p$ and cluster centroid $\overline{D}$. Euclidean distance as calculated by the distance function represents the associated information loss. The pseudocode for the *dissolve* operation follows:

```
dissolve(P) {
    C ← some cluster of P;
    for each point p ∈ C
        D ← some cluster of D ∈ P \ {C}  minimizing  δ(p, D̄);
        D ← D ∪ {p};
    return P \ {C};
}
```

The *distill* operation maintains feasibility as it increases the number of clusters in the partition. Clusters that contain excess points are referred to as oversized. $S$ is the set of oversized clusters. $Q$ is the set of all excess points. $N$ is the new cluster. So, when distilling $N$ an excess point $p$ is selected at random from the set of excess points $Q$. This is used to seed $N$. The point $p$ is removed from $Q$ and shifted from the oversized cluster $C_p$ where it resides to the new cluster $N$. The centroid for $N$ is the new point. While the first point was selected at random all subsequent points are selected to minimize information loss (Euclidean distance) of $N$. The closest point $q$ to $N$ is removed from $Q$, shifted from cluster $C_q$ to $N$, and the centroid is recalculated for $N$. This is repeated until the new cluster $N$ is filled with *k* points. The pseudocode for *distill* follows:

```
distill(P) {
    S ← set of clusters C ∈ P such that |C| > k;
    Q ← excess points of the oversized clusters S;
    p ← some point of Q;
    Q ← Q \ {p};
    N ← {p};
    while (|N| < k)
        p ← some point q ∈ Q that minimizes δ(q, N̄);
        Q ← Q \ {p};
        Cp ← Cp \ {p};
        N ← N ∪ {p};
    return P ∪ {N};
}
```

The acceptance criterion is used as a tuning parameter that lets a new local optimum be used in the search path even though it is not better than the current best-found one. It can be advantageous to use the inferior of the two in the next iteration. Doing so adds diversification to the search path aiding escape from the current local optimum. The acceptance criterion decides which of the current best or new inferior solution to use in the next iteration. When an inferior local optimum is chosen, it is said to bias search toward diversification. The intention is to add enough diversity that leads to a breakout. If after some number of explorative cycles, it does not find a new best solution, the acceptance criterion rejects the current inferior solution and returns the search back to the current best solution. The pseudocode for the acceptanceCriterion follows:

```
acceptanceCriterion(P, bestP) {
    if SSE(P) < SSE(bestP) return P;
    random ← Uniform random real number in [0,1];
    if (random < A) return P"; // A = 0.8 in original work
    else return bestP;
}
```

**Avoiding Costly Tests with ILSMC**

LSC and *updateC* avoid vast numbers of tests by targeting clusters to test within modified versions of LS and *update*. The approach is effective because few clusters interact in most of the calls to *update*. How LSC and *updateC* work to avoid vast number of tests is discussed above. A naïve version of ILSMC would just replace LS($P'$) with LSC($P'$, $P'$) inside ILSM. While experiments showed significant improvement for naïve ILSMC compared to ILSM, an opportunity to avoid even more tests would be missed.

The calls to the *distill* and *dissolve* perturbation operations in naïve ILSMC are sandwiched between calls to LSC. The perturbances are very localized within these operations. This results in very few clusters being changed by the perturbations thus giving opportunities for exploitation within LSC. If the changed clusters are tracked within both perturbation operations, this could be communicated to LSC and then to *updateC*. If there is not a target set of clusters passed in, LSC must be called with the complete set of cluster pairs. By communicating a target set of clusters to LSC, vast numbers of additional cluster pairs can be avoided. This is in comparison to LSC without any form of communications from the perturbation operations. In summary, tracking the changed clusters within the perturbation operations, *dissolveC* and *distillC*, and communicating them as targeted clusters to LSC allows many more tests to be avoided compared to naïve ILSMC. Pseudocode for *dissolveC* and *distillC* follow.

```
dissolveC(P) {
    C ← some cluster of P;
    changed ← ∅;                          // <-- new line
    for each point p ∈ C
        D ← some cluster of D ∈ P \ {C} minimizing δ(p, D̄);
        D ← D ∪ {p};
        changed ← changed ∪ {D};       // <-- new line
    return (P \ {C}, changed);           // <-- modified
}

distillC(P) {
    S ← set of clusters C ∈ P such that |C| > k;
    Q ← excess points of the oversized clusters S;
    p ← some point of Q;
    Q ← Q \ {p};
    N ← {p};
    changed ← {N};                        // <-- new line
    while (|N| < k)
        p ← some point q ∈ Q that minimizes δ(q, N̄);
        Q ← Q \ {p};
        C_p ← C_p \ {p};
        N ← N ∪ {p};
        changed ← changed ∪ {C_p};     // <-- new line
    return (P ∪ {N}, changed);           // <-- modified
}
```

The changes to *dissolve* and *distill* are minimal. Two lines are added, and one modified in both *dissolveC* and *distillC* (see source code lines commented with "*new line*" and "*modified*"). The first *new line* in each operation initializes the *changed* variable. The second *new line* adds changed clusters to the *changed* set. The last line in both perturbation operations is modified to return a tuple which includes the perturbed $k$-partition and the set of changed clusters to be targeted. Now ILSMC can be created with minimal changes to ILSM, just replace LS, *perturb*, *dissolve*, and *distill* procedures with LSC, *perturbC*, *dissolveC*, and *distillC*. The pseudocode for ILSMC follows.

```
ILSMC(P) {
    targetClusters ← {C | C∈P}                    // <-- new line
    P ← LSC(P, targetClusters);                   // <-- modified
    bestP ← P;
    while (not terminationCondition)
        (P’, targetClusters) ← perturbC(P);   // <-- modified
        P” ← LSC(P’, targetClusters);         // <-- modified
        P ← acceptanceCriterion(P”, bestP);
        If (SSE(P) < SSE(bestP))
            bestP ← P;
    return bestP;
}

perturbC(P) {
    if (size(P) = minP) return distillC(P);
    else if (size(P) = maxP) return dissolveC(P); // <-- modified
    else if (random < 0.5) return distillC(P);    // <-- modified
    else return dissolveC(P);                     // <-- modified
}
```

**Using Sampling to Bias Dissolve**

During this study it was observed that perturbations tend to find larger reductions in information loss when they involve clusters with higher information loss. The *dissolve* operation uses a uniform probability distribution to pick the clusters to *dissolve* and places no focus on relative information loss. This study considered a few new probability distributions biased toward selecting clusters with higher information loss. Preliminary experiments with sloped straight-line probability distributions biased toward clusters with higher information loss were beneficial. To apply the new probability distributions, clusters need to be sorted by increasing amount of information loss. The $i^{th}$ cluster where $i \geq 1$ and $i \leq |P|$ is then selected with the probability defined by the probability mass function. A couple probability mass functions were developed but the following probability mass function was settled upon and used in the experiments:

$X$ is a discrete random variable with range $x$

$$f(x) = \mathrm{P}(X = x) = \frac{1}{|P|^s}(x^s - (x-1)^s)$$

$x \in \{1, 2, \dots, |P|\}$

$s \in \mathbb{Z}^+$

The variable $s$ is the sampling constant and $x$ is the ordinal number of the $i^{th}$ cluster in the set of clusters $P$ ordered from lowest information loss to the highest. However, to select clusters with probabilities defined by this probability mass function does not require the set of clusters to be sorted by information loss before every selection. This is very advantageous since it is costly to sort the clusters within the *dissolve* operation. Instead several cluster samples are selected at random from the *k*-partition and the one with the largest information loss is selected. There is the possibility of selecting a cluster more than once. The constant $s$ is the sampling constant and corresponds to the number of samples selected. The probability that $x$ the $i^{th}$ cluster is selected is $\mathrm{P}(X = x)$ defined by the probability mass function above. Notice that when the sampling constant is set to 1 the probability mass function defines the uniform probability distribution. Increasing the sampling constant increases the bias toward clusters with higher information loss. The sampling constant is a tuning parameter for increasing intensification within *dissolve*. A good value for the sampling constant was identified and set to $s = 5$ for the experiments. The pseudocode for *bDissolve* follows:

```
bDissolve(P, s) {
    S ← s random clusters sampled C_i ∈ P for i = [1, s];
    C ← cluster with the highest information loss from S;
    changed ← ∅;
    for each point p ∈ C
        D ← some cluster D \ {C} minimizing δ(p, D̄);
        D ← D ∪ {p};
        changed ← changed ∪ {D};
    return (P \ {C}, changed);
}
```

Bias will not be introduced into the *distill* operation. Preliminary experiments suggested that biasing the selection of a starting point is not yet decisively beneficial. First, there are relatively few excess points. Second, there are only weak correlating characteristics currently identified that facilitate selection of points with higher tendencies toward better outcomes. Further work needs to be done on ways to identify better starting points.

## A Dynamic Acceptance Criterion

Search is generally the process of iteratively moving from the current best solution to better solutions. However, iterative local search through the acceptance criteria employs a strategy that allows interim search moves to solutions of worse quality. The intention is to add diversification to aid escape from local optima. The concept is that a few worse moves will lead to a breakout and to a better local optimum. As described earlier, the acceptance criterion is a key component of iterated local search. If the new solution in not the current best-found solution, it decides which of two partitions to process next. Accepting the new inferior solution P or rejecting it and returning to the current best-found solution instead. In ILSM the acceptance criterion is a fixed probability $A$. Higher values of $A$ emphasize diversification and lower values intensification. A good fixed value for $A$ was found to be 80% (Laszlo & Mukherjee, 2015). The formulas that follow are the probabilities of selecting $P$ rather than $bestP$ under each of two possible conditions:

$$\Pr(choosing\ P\ over\ bestP) = \begin{cases} 1, & \text{SSE}(P") < \text{SSE}(bestP) \\ A, & \text{otherwise} \end{cases}$$

This acceptance criterion implements a uniform probability distribution. The pseudocode follows:

```
acceptanceCriterion(P, bestP) {
    if SSE(P) < SSE(bestP) return P;
    random ← Uniform random real number in [0,1);
    if (random < A) return P”; // A = 0.8 in original work
    else return bestP;
}
```

The improvement in this section modifies the acceptance criterion above. The objective is to accept current solutions closer in quality to the best-found solution with higher probability than ones with lesser quality. The modification changes the acceptance probability to a formula based on the difference in error between $P$ and the current best solution. The idea is to vary acceptance probability based on the size of the increase in the sum of the squared errors. The formulas that follow are the new probabilities of selecting $P$ rather than $bestP$ under each of two possible conditions:

$$\Pr(\text{choosing } P \text{ over } bestP) = \begin{cases} 1, & \text{SSE}(P”) < \text{SSE}(bestP) \\ e^{\left(\frac{(\text{SSE}(bestP) - \text{SSE}(P"))}{(T * \text{SSE}(bestP))}\right)}, & \text{otherwise} \end{cases}$$

The pseudocode for the new acceptance criterion follows:

```
dAcceptanceCriterion(P", bestP) {
    if (SSE(P") < SSE(bestP)) return P";
    random ← Uniform random real number in [0,1];
    if (random < e^(((SSE(bestP) - SSE(P"))) / (T * SSE(bestP)))) return P";
    else return bestP;
}
```

If the difference in errors between the current best-found solution and $P$ is relatively small, then the probability of acceptance is adjusted to be high. If the difference in errors is relatively high, the probability of acceptance is adjusted to be low. The constant $T$ is a tuning parameter, it was experimentally found to work best at 0.00001. By varying the acceptance criterion, it can quickly reject a path with increasing SSE since the likely benefit of continuing the search is decreasing. Otherwise, it will keep accepting inferior solutions as long as the error remains low enough and there is still a relatively higher likelihood of benefit in continuing the search.

# Chapter 4

# Results

This chapter presents results from experiments using benchmark datasets. The results demonstrate performance of LSC, ILSMC, ILSM with *bDissolve*, and ILSM with *dAcceptanceCriterion* compared to the original LS and ILSM. The results that follow answer the three research questions posed in Chapter 1 using the methodology in Chapter 3.

## Introduction

Laszlo and Mukherjee (2015) presented LS and ILS for microaggregation (ILSM) for producing *k*-anonymity microaggregations. They demonstrated their algorithm has advantages over extant microaggregation methods. The goal of this study was to demonstrate the advantages of three novel improvements to LS and ILSM. The first improvement adds cluster tracking to LS, ILSM and the original perturbation operations to create LSC and ILSMC. The second improvement adds biasing to the *dissolve* perturbation operation and creates *bDissolve*. It biases the perturbation operator toward clusters with higher loss. The third improvement changes the acceptance criteria from static to dynamic creating *dAcceptanceCriteria*. It dynamically changes the probability of acceptance based on the difference in quality between the current solution and the best solution found so far. Current solutions closer in quality to the best solution are accepted with higher probability than ones with lesser quality.

## Benchmark Datasets

The experiments used the following three benchmark datasets: *Tarragona* (834 records with 13 attributes), *Census* (1082 records with 13 attributes), and *EIA* (4092 records with 10 attributes). These benchmarks were also used in the Laszlo and Mukherjee (2015) study and previous studies. As in Laszlo and Mukherjee (2015), the dataset attributes were normalized to a mean of zero and a standard deviation of one so that no single attribute would have a disproportionate effect on the results from the experiments. Table 2 presents the first row of data from each of the three datasets.

| | *Tarragona* | *Census* | *EIA* |
|---|---|---|---|
| Attribute 1 | -0.37861 | 0.739487803 | -0.427604166 |
| Attribute 2 | -0.48639 | -0.432373901 | -0.538286816 |
| Attribute 3 | -0.2837 | 0.713255797 | -0.32878662 |
| Attribute 4 | -0.02597 | -0.596032084 | -0.458881554 |
| Attribute 5 | -0.02308 | 0.013888974 | -0.539673926 |
| Attribute 6 | 0.07261 | -0.640145244 | -0.612108827 |
| Attribute 7 | -0.068 | -0.419519668 | -0.061995332 |
| Attribute 8 | -0.41123 | -0.543431262 | -0.196206166 |
| Attribute 9 | -0.1062 | -0.371754498 | -0.430195776 |
| Attribute 10 | -0.03452 | 0.260923742 | -0.574804969 |
| Attribute 11 | -0.00071 | 0.362487702 | - |
| Attribute 12 | -0.14978 | 0.290109471 | - |
| Attribute 13 | -0.13883 | 0.341212348 | - |

**Table 2:** First row of data from each dataset

**Experimental Results**

As with the prior study, this study ran experiments on partitions with the same values of $k$ ($k$ = 3,4,5,6 and 10). This study also uses the standardized information loss described in Section 3 as the measure of quality for the $k$-partition. All the experiments ran Java bytecode. All experiments were performed on 2.3 GHz Intel Core i7-3615QM

This study recreated the experiments published by Laszlo and Mukherjee (2015). It achieved similar results in terms of quality (percentage loss of information) and execution times. Figures are not shown for values of $k$ = 4 and $k$ = 6 because the results for $k$ = 3 and $k$ = 5 are representative for those values of $k$ in all test cases.

**LSC versus LS**

Experiments consisted of 5000 runs of LSC and LS each in a random restart regime. Quality in terms of percentage loss of information and execution times were recorded. The averages for the 5000 runs were computed and used to compare LSC and LS.

Table 3 reports the average execution time per run in seconds for LSC and LS. The numbers in parentheses present the ratio of LS to LSC run times. Table 4 reports the average results for quality (percentage information loss) for each experiment test case.

Figure 1 through Figure 3 show the dramatic decrease in sizes of the cluster pair lists with iterations of *updateC* as compared to *update*. Within both LSC and LS is a loop that calls (iterates) *updateC* and *update*, respectively. At the beginning of both *updateC* and *update*, a new list of cluster pairs is constructed. The sizes of the cluster pair lists for each of the $n^{th}$ iterations were recorded. The sizes were then averaged per the $n$ iterations over the 5000 runs and charted for comparison purposes. While 15 charts were constructed for the three datasets, one for each of the 5 values of $k$, the value of $k$ had minor impact on the shape of the charts. The three figures are representative of all values of *k*.

| | k = 3 | k = 4 | k = 5 | k = 6 | k = 10 |
|---|---|---|---|---|---|
| *Tarragona* | | | | | |
| LSC run | 0.037 | 0.052 | 0.074 | 0.094 | 0.16 |
| LS run | 0.063 (1.70) | 0.086 (1.65) | 0.12 (1.62) | 0.150 (1.60) | 0.23 (1.44) |
| *Census* | | | | | |
| LSC run | 0.040 | 0.044 | 0.054 | 0.066 | 0.12 |
| LS run | 0.070 (1.75) | 0.071 (1.61) | 0.084 (1.56) | 0.100 (1.52) | 0.18 (1.5) |
| *EIA* | | | | | |
| LSC run | 0.69 | 0.44 | 0.35 | 0.31 | 0.30 |
| LS run | 1.43 (2.07) | 0.85 (1.93) | 0.59 (1.69) | 0.51 (1.65) | 0.49 (1.63) |

**Table 3:** Average execution time per run (in seconds): LSC compared to LS

The average execution time for LSC was significantly reduced compared to LS (see Table 3). The table shows the greatest reduction was 52% for the *EIA* dataset and $k = 3$. The least reduction was 30% for the *Tarragona* dataset and $k = 10$.

The results charted in Figure 1 through Figure 3 shows why there is a significant reduction in execution times. Note from Figure 1, every run of LSC had 18 or fewer iterations of *updateC* for the *Tarragona* dataset and $k = 3$. LS had 22 or fewer iterations of *update*. The cluster pairs list sizes averaged less than 1500 pairs or fewer for more than half the iterations with LSC and *updateC* compared to LS and *update* with average list sizes of 35K+ for all but the last several iterations. Looking at Figure 1 through Figure 3 most cluster pairs for most iterations are not in the cluster pair lists thus avoiding the associated quick reject tests, shift tests and swap tests (i.e. *mayShift*, *maySwap*, *shiftTest* and *swapTest*). By the seventh iteration the number of cluster pairs eliminated from the cluster pairs lists of LSC were greater than 75%, 80% and 85% respectively for the

*Tarragona*, *Census*, and *EIA* datasets. An obvious question is why only a 30% to 53% reduction in execution times given the vast number of cluster pairs eliminated from the lists. The reason is that many of the associated tests that were avoided are only the quick reject tests which are already very efficient and fast. The experiments demonstrated the advantages of LSC over LS in terms of execution run times.

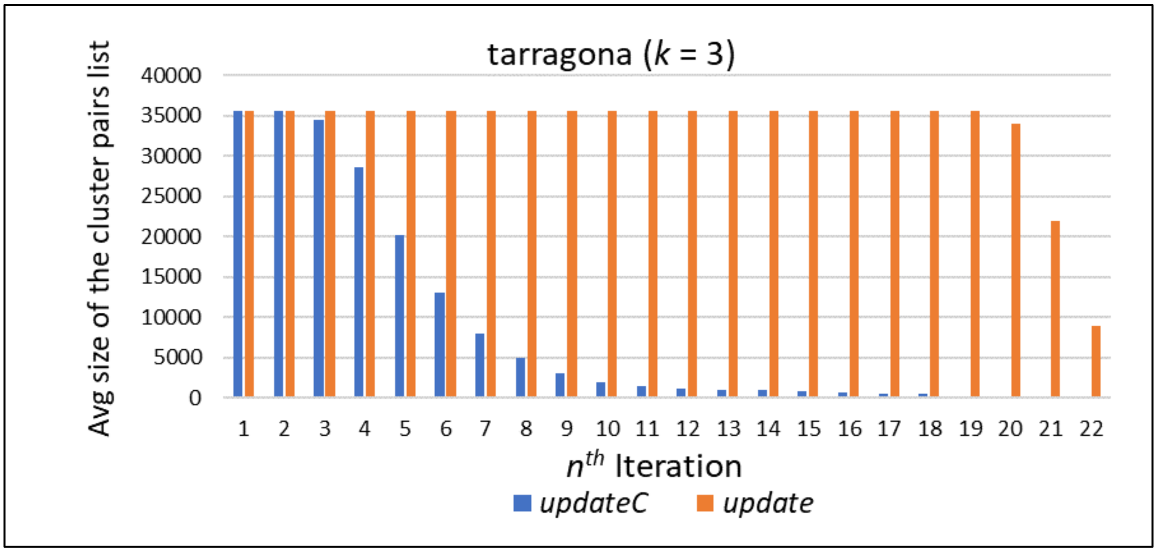


**Figure 1:** Decrease in cluster pairs list size with *updateC* iterations (*Tarragona* 3)

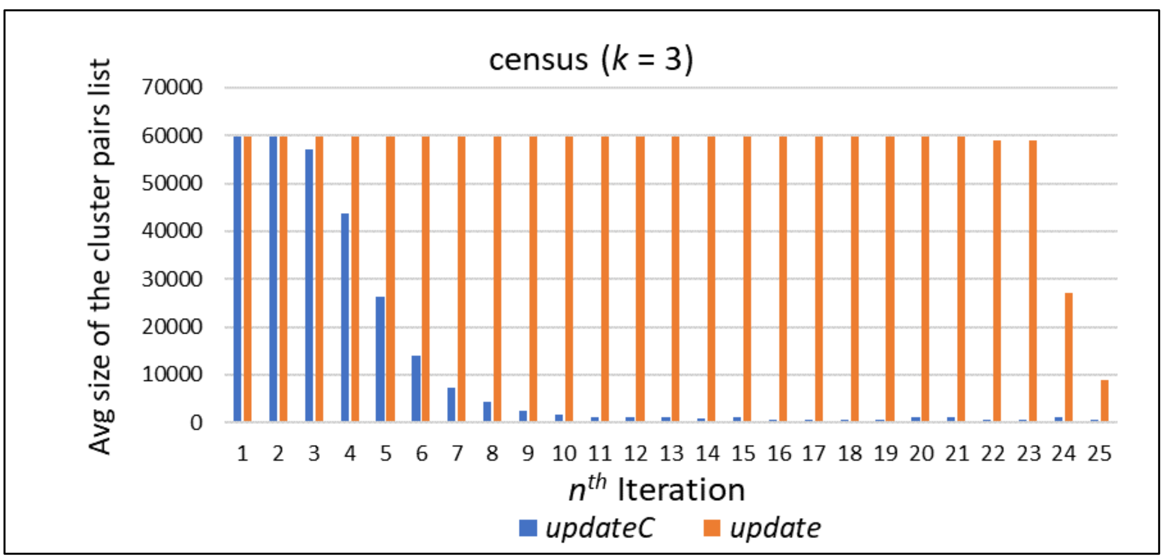


**Figure 2:** Decrease in cluster pairs list size with *updateC* iterations (*Census* 3)

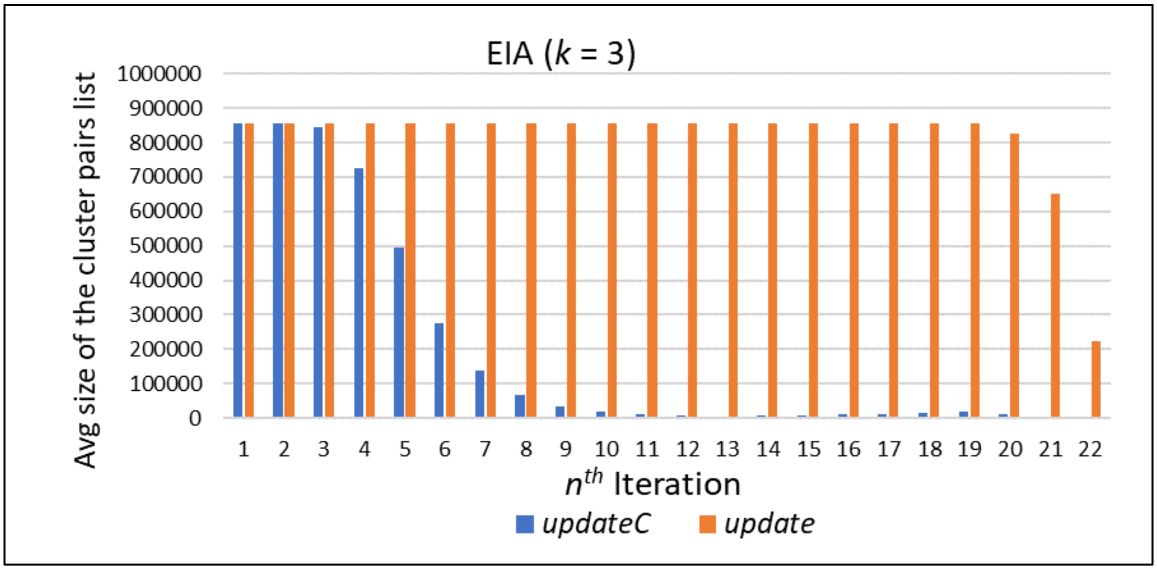


**Figure 3:** Decrease in cluster pairs list size with *updateC* iterations (*EIA* 3)

The average results for quality (information loss) were similar for both LSC and LS (see Table 4). On a per iteration basis, the results did not show LSC converging faster than LS. Faster convergence of LSC as compared to LS was entirely due to shorter iteration execution times helped by the improved computational efficiency. This was not unexpected because *updateC* only rejects cluster pairs that are assured to fail the shift and swap tests. It is just a quicker reject test that complements *mayShift* and *maySwap*.

| | k = 3 | k = 4 | k = 5 | k = 6 | k = 10 |
|---|---|---|---|---|---|
| *Tarragona* | | | | | |
| LSC Best | 14.68% | 17.23% | 20.32% | 23.66% | 30.23% |
| LS Best | 14.68% | 17.24% | 20.30% | 23.66% | 30.22% |
| *Census* | | | | | |
| LSC Best | 4.87% | 6.67% | 7.88% | 8.86% | 11.96% |
| LS Best | 4.86% | 6.66% | 7.87% | 8.86% | 11.94% |
| *EIA* | | | | | |
| LSC Best | 0.44% | 0.59% | 1.21% | 1.02% | 2.46% |
| LS Best | 0.45% | 0.59% | 1.19% | 1.03% | 2.45% |

**Table 4:** Information loss: LSC compared to LS after 5000 runs

**ILSMC versus ILSM**

Experiments consisted of 20 runs of ILSMC and ILSM where each run terminated after 5000 iterations. The runs were started with random *k*-partitions. Quality in terms of percentage loss of information and execution run times were recorded. The averages for the 20 runs were computed and used to compare ILSMC to ILSM.

Figure 4 through Figure 8 show how average information loss decreases over time for ILSMC compared to ILSM across a representative set of results. The blue lines are for ILSMC and the orange lines are for ILSM. ILSMC ran significantly faster than ILSM and it why the blues lines are much shorter. The most dramatic results were for the *EIA* dataset, *k* = 3, where ILSMC ran in 6.01 seconds compared to 1371 seconds for ILSM. The effects on computational efficiency lessen as the values for *k* get larger but still the least speedup was a speed up of 4 times for *Tarragona*, *k* = 10.

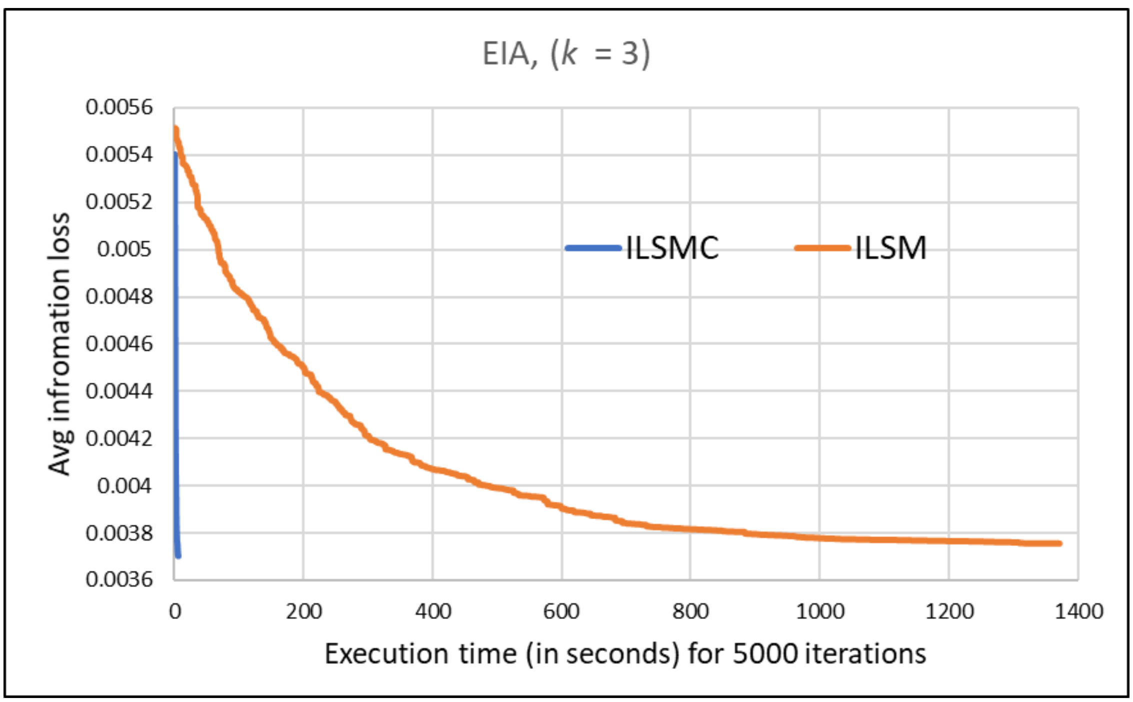


**Figure 4:** Decrease in information loss with time, ILSMC vs ILSM (*EIA* 3)

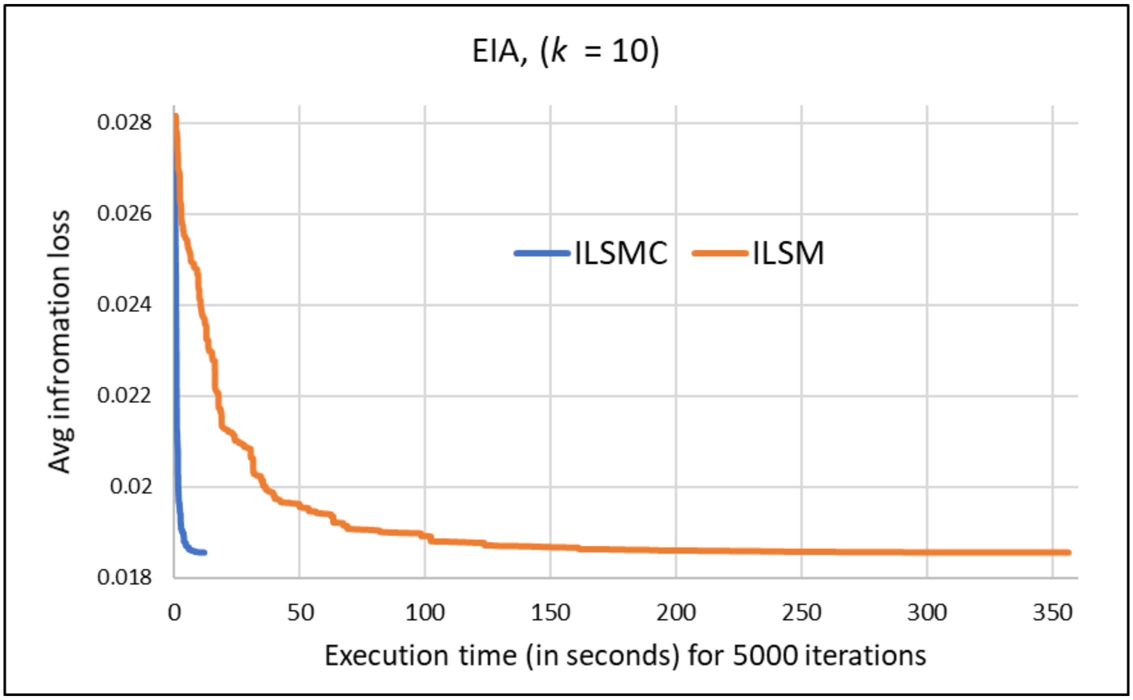


**Figure 5:** Decrease in information loss with time, ILSMC vs ILSM (*EIA* 10)

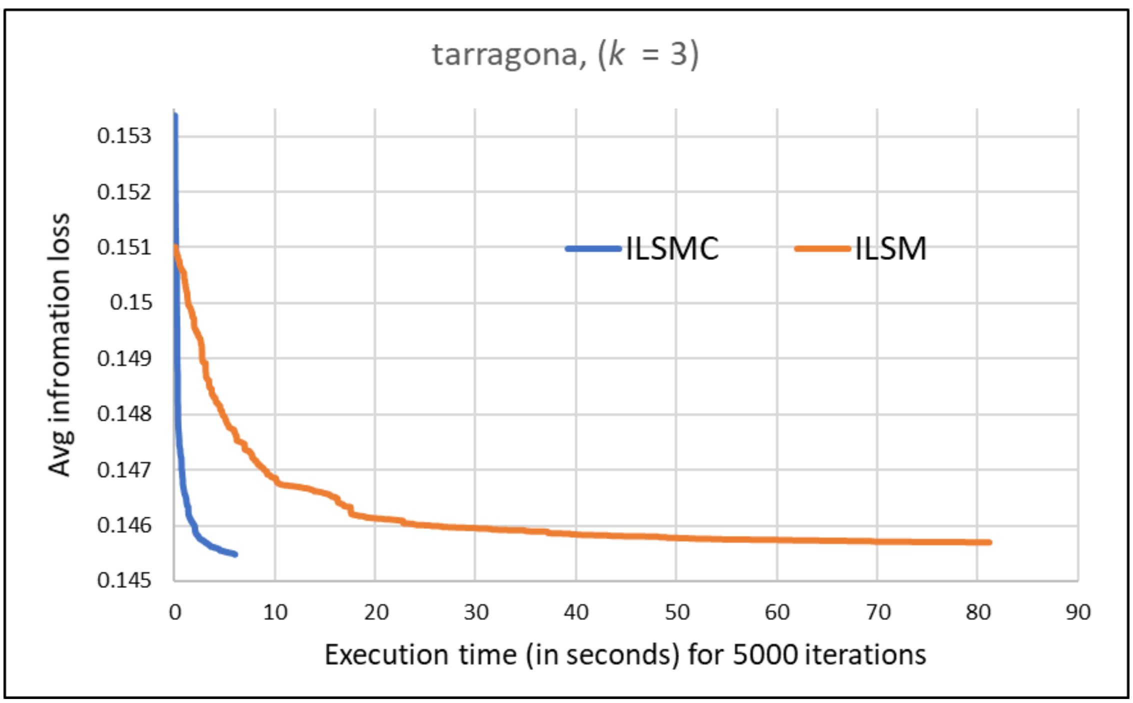


**Figure 6:** Decrease in information loss with time, ILSMC vs ILSM (*Tarragona* 3)

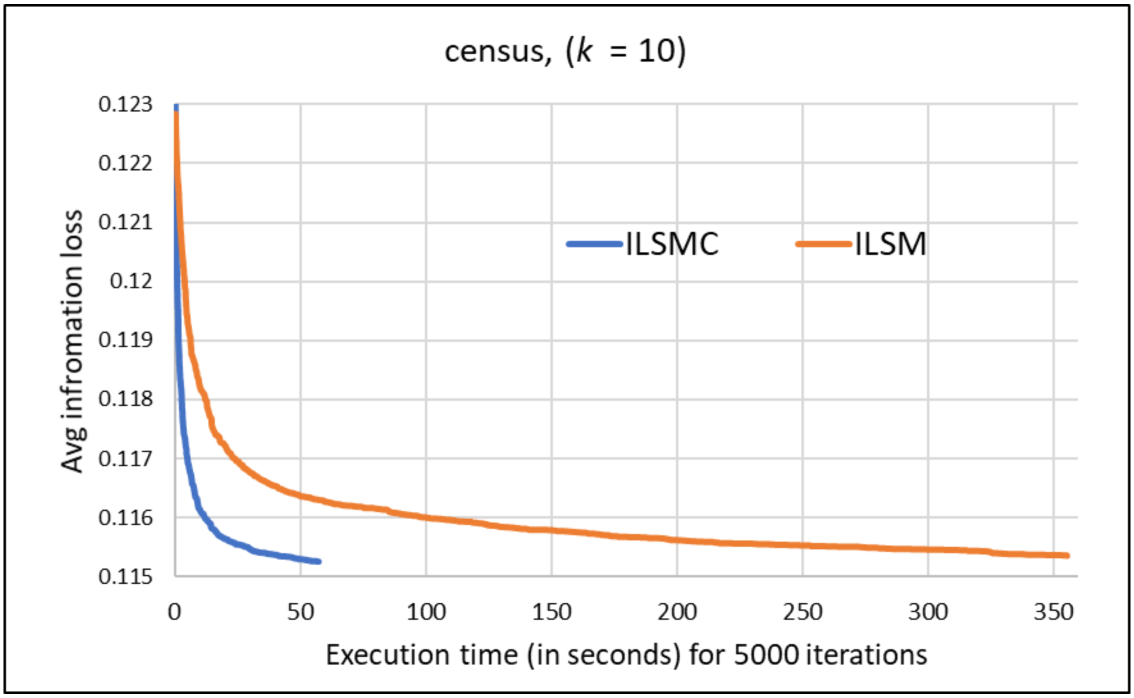


**Figure 7:** Decrease in information loss with time, ILSMC vs ILSM (*Census* 10)

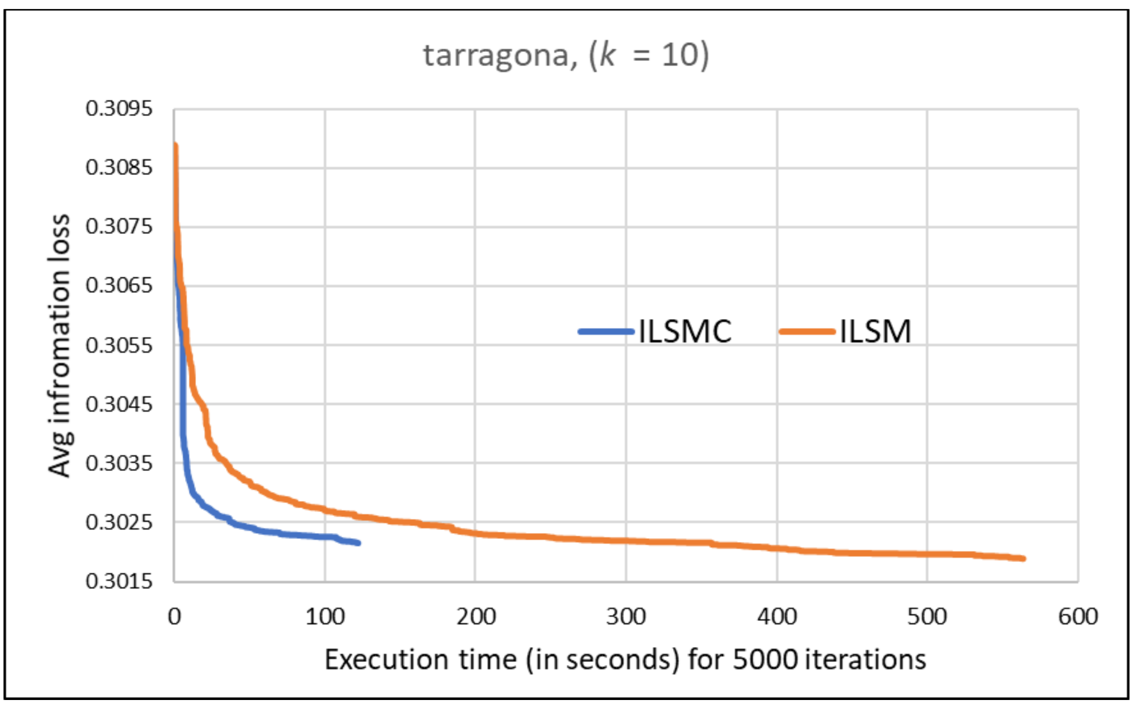


**Figure 8:** Decrease in information loss with time, ILSMC vs ILSM (*Tarragona* 10)

Table 5 reports the average execution elapsed time per iteration in seconds for ILSMC and ILSM. The numbers in parentheses present the total execution time for 5000 iterations. The average times for ILSMC were significantly reduced compared to ILSM. Again, the greatest reduction in time was for the *EIA* dataset and $k = 3$. The average time for ILSMC runs with this dataset were 6.01 seconds compared to 22 minutes and 51 seconds for ILSM runs, a reduction of 99.6%. The least reduction in time was for the *Tarragona* dataset and $k = 10$, where the average time for ILSMC was 2 minutes and 2 seconds compared to 9 minutes and 24 seconds for ILSM runs, a reduction of 78%. The *Census* dataset saw reductions from 84% to 96% as compared to ILSM.

| | k = 3 | k = 4 | k = 5 | k = 6 | k = 10 |
|---|---|---|---|---|---|
| *Tarragona* | | | | | |
| ILSMC iteration | 0.0011 (5.73) | 0.0021 (10.5) | 0.004 (19.8) | 0.0069 (34.3) | 0.024 (122) |
| ILSM iteration | 0.016 (81) | 0.025 (123) | 0.039 (193) | 0.055 (275) | 0.11 (564) |
| *Census* | | | | | |
| ILSMC iteration | 0.00070 (3.49) | 0.00094 (4.7) | 0.0017 (8.5) | 0.0027 (13.6) | 0.011 (57.3) |
| ILSM iteration | 0.018 (90) | 0.017 (84.5) | 0.023 (114) | 0.03 (151) | 0.071 (356) |
| *EIA* | | | | | |
| ILSMC iteration | 0.0012 (6.01) | 0.0011 (5.26) | 0.0011 (5.41) | 0.0012 (5.95) | 0.0024 (11.8) |
| ILSM iteration | 0.27 (1371) | 0.14 (689) | 0.089 (445) | 0.069 (346) | 0.071 (356) |

**Table 5:** Average execution time (in seconds) per iteration: ILSMC compared to ILSM

Table 6 compares the quality of solutions recorded using ILSMC compared to those recorded by ILSM. The average results for quality (percentage information loss) were nearly identical for both ILSMC and ILSM. In Chapter 3, there was a concern that slightly lifting the requirement for randomness of the cluster pair lists could have a negative effect on quality. No negative effects were seen, but neither did the results show ILSMC converged faster than ILSM on a per iteration basis. Faster convergence was entirely due to the shorter iteration times. The results show that quality was not

| | k = 3 | k = 4 | k = 5 | k = 6 | k = 10 |
|---|---|---|---|---|---|
| *Tarragona* | | | | | |
| ILSMC Best | 14.50% | 17.12% | 20.17% | 23.52% | 30.14% |
| ILSMC Avg | 14.55% | 17.15% | 20.19% | 23.58% | 30.22% |
| ILSMC Worst | 14.58% | 17.19% | 20.22% | 23.65% | 30.30% |
| ILSM Best | 14.48% | 17.11% | 20.17% | 23.52% | 30.14% |
| ILSM Avg | 14.57% | 17.15% | 20.21% | 23.59% | 30.19% |
| ILSM Worst | 14.60% | 17.20% | 20.28% | 23.65% | 30.29% |
| *Census* | | | | | |
| ILSMC Best | 4.77% | 6.13% | 7.37% | 8.34% | 11.46% |
| ILSMC Avg | 4.81% | 6.20% | 7.44% | 8.41% | 11.53% |
| ILSMC Worst | 4.86% | 6.29% | 7.50% | 8.47% | 11.62% |
| ILSM Best | 4.77% | 6.13% | 7.38% | 8.33% | 11.47% |
| ILSM Avg | 4.80% | 6.19% | 7.44% | 8.41% | 11.54% |
| ILSM Worst | 4.83% | 6.30% | 7.53% | 8.50% | 11.69% |
| *EIA* | | | | | |
| ILSMC Best | 0.37% | 0.51% | 0.76% | 0.94% | 1.85% |
| ILSMC Avg | 0.37% | 0.52% | 0.81% | 0.96% | 1.86% |
| ILSMC Worst | 0.38% | 0.53% | 0.98% | 0.99% | 1.86% |
| ILSM Best | 0.37% | 0.52% | 0.76% | 0.94% | 1.85% |
| ILSM Avg | 0.38% | 0.52% | 0.82% | 0.95% | 1.86% |
| ILSM Worst | 0.41% | 0.54% | 0.96% | 0.97% | 1.86% |

**Table 6:** Information loss using: ILSMC compared to ILSM

significantly affected by the cluster change tracking within ILSMC. Again, this is not unexpected since change tracking does not affect the mechanism for selecting next steps in the search path (i.e. swaps and shifts).

For every call to LSC and LS, the sizes of the cluster pair lists within each of the nth iterations of *updateC* and *update* were recorded. The sizes were then averaged per the *n* iterations over the 20 runs of ILSMC and ILSM and charted for comparison purposes. While 15 charts were constructed for the three datasets, one for each of the 5 values of *k*, the value of *k* had minor impact on the overall shape of the charts. The charts in Figure 9 through Figure 11 are representative of the three datasets for all values of *k*. The Figures show how the decrease in the size of cluster pair lists greatly increases computational efficiency.

The results charted in Figure 9 through Figure 11 show why there were significant reductions in execution times. The following describes how to interpret those charts. ILSMC calls LSC 5001 times during a run. The first call to LSC takes a random *k*-partition and finds a current best solution. This is the initialization call and it only happens once per run. The left blue bars describe this initial call in terms of average size of cluster pair lists per iteration of *updateC*. It is similar in shape to Figures 1 through 3 in the discussion about LSC vs LS. Each bar is the average of 20 values (20 runs, one call to LSC per run). The middle orange bars describe all the calls to *updateC* within the calls to LSC which follow a perturbation operation. Each bar is the average of 100K values (20 runs, 5000 calls per run). The gray bars on the right describe all the calls to *update* within LS and ILSM (20 runs, 5001 calls per run). The size of cluster pair lists is fixed within LS; however, in Figure 9 through Figure 11 the gray bars decrease for the last several iterations. This is because not all calls to *update* went the full 22 iterations. When a call went for example, 20 iterations, then a zero instead of 35K went into calculating the averages for iterations 21 and 22.

It is important to note the larger charts with the blue, orange, and gray bars are charted with a logarithmic scale for the vertical axis. The logarithmic scale is needed to better illustrate the enormous difference in sizes for the cluster pair lists and better illustrate the lower values. The inset charts are just the same orange bars for *updateC* (perturbations) but with a linear scale for the vertical axis. The inset chart better illustrates the diversity in list sizes over the successive iterations of *updateC* (perturbations) which are masked by the logarithmic scale. The use of both scales is for clarity.

For *updateC* the sizes of cluster pair lists were nearly 2 orders of magnitude less than for *update*, except for the first initialization call to *updateC*. (see Figure 9 through Figure 11). This resulted in vast numbers of associated tests (i.e. *mayShift, maySwap, shiftTest* and *swapTest*) being avoided. When comparing Figure 1 through Figure 3 to Figure 9 through Figure 11, note the sizable advantage that ILSMC has over running LSC in a random restart regime. Every iteration of LSC in the random restart experiment started with a complete cluster pairs list compared to ILSMC where the first call to *updateC* was the only one. All subsequent calls to *updateC* within ILSMC were started with small cluster pair lists. This resulted because the perturbation operations make relatively small localized changes affecting only a small number of clusters. This difference is why ILSMC combined with LSC had a greater impact on execution times compared to LSC alone.

Figure 9 (*EIA* dataset, $k = 3$) shows average list size was never greater than 4500 for *updateC* and many were much smaller. That compared to most iterations with nearly 850K cluster pairs for *update*. Figure 10 (*Tarragona* dataset, $k = 3$) shows cluster pair lists within *updateC* were never larger than 1500 pairs. This compared to *update* which processed 35K pairs on average for all but the last few iterations. Figure 11 (*Census* dataset, $k = 3$) shows average list size for *updateC* was never larger than 1900 pairs. This compared to 60K cluster pairs for most *update* iterations.

The results demonstrate that ILSMC is highly effective at reducing sizes of cluster pair lists resulting in most of the quick reject, swap and shift tests being avoided. The result is greater computational efficiency in ILSMC as compared to ILSM.

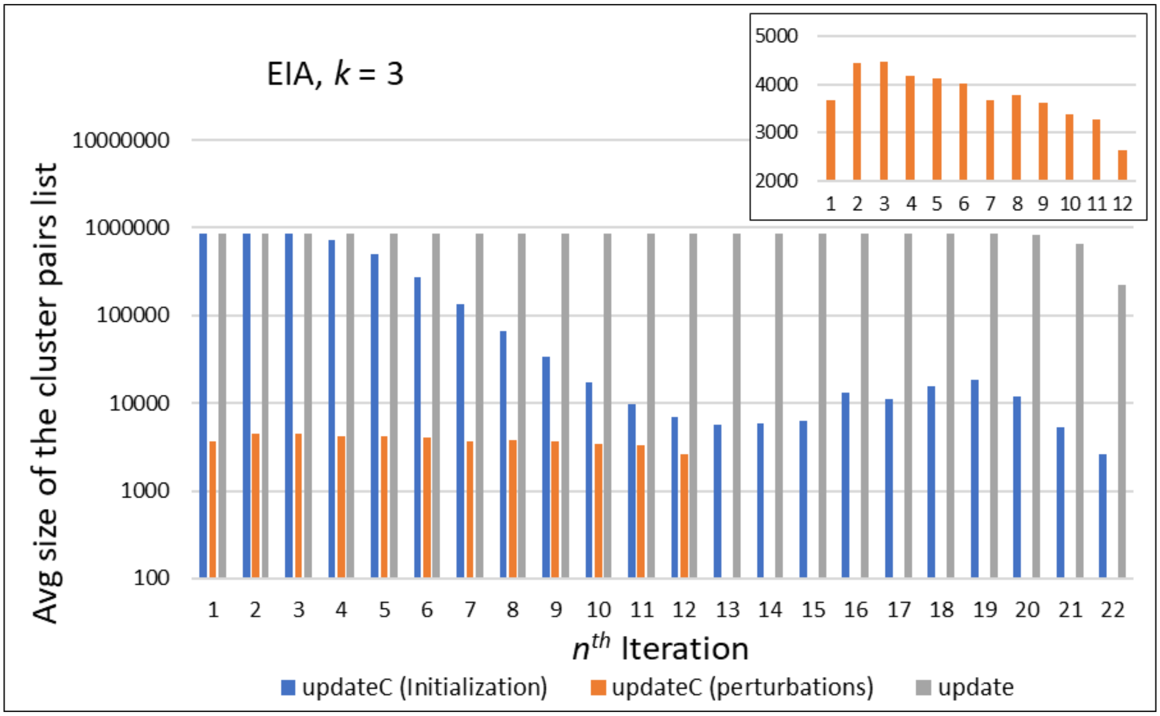


**Figure 9:** Decrease in cluster pairs list size with *updateC* (*EIA* 3)

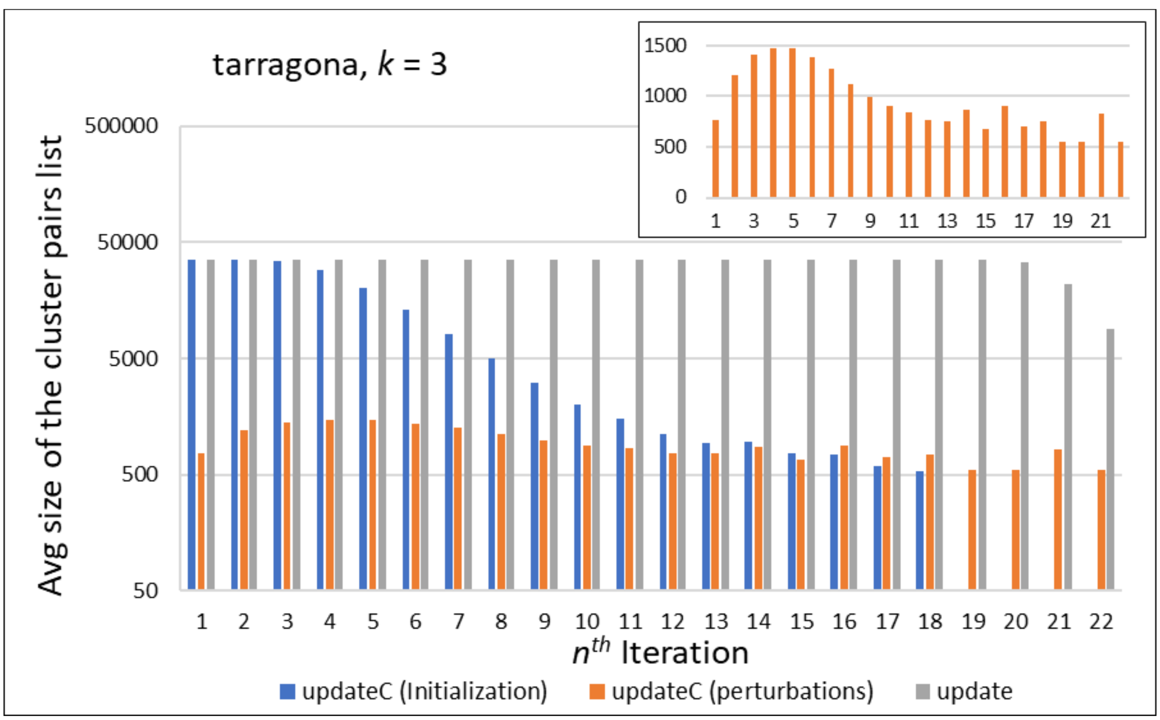


**Figure 10:** Decrease in cluster pairs list size with *updateC* (*Tarragona* 3)

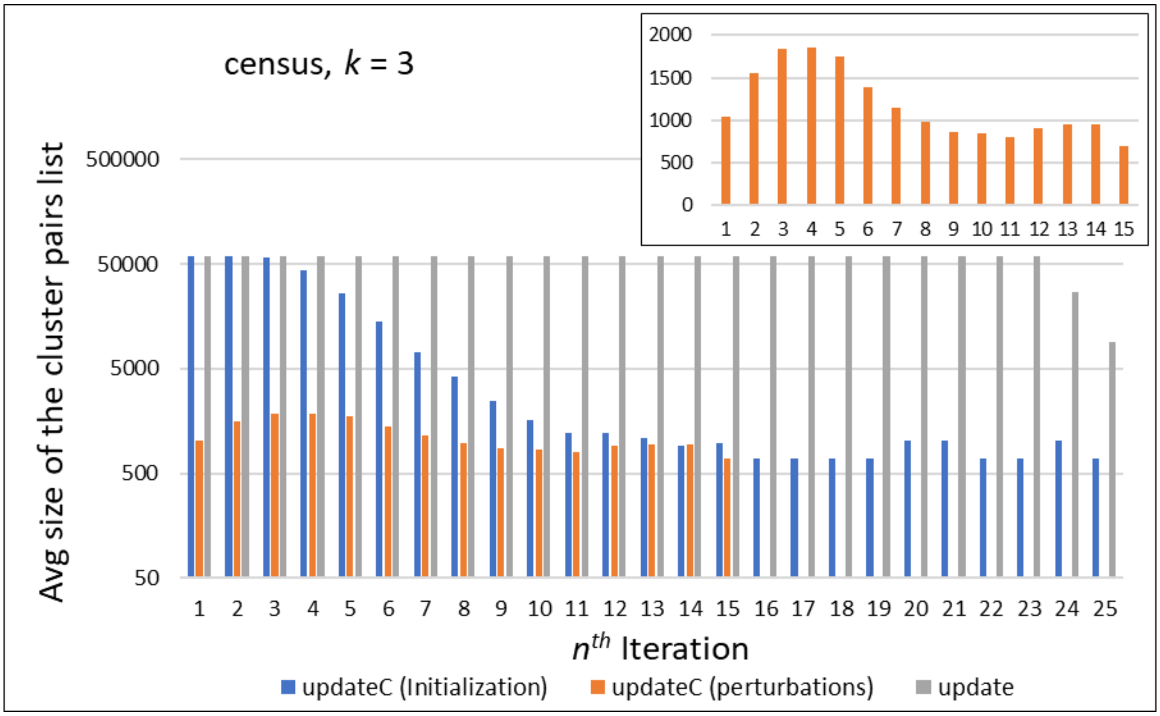


**Figure 11:** Decrease in cluster pairs list size with *updateC* (*Census* 3)

### *bDissolve* versus *dissolve*

Experiments consisted of 20 runs of ILSM where each run terminated after 5000 iterations. The *dissolve* perturbation operation within ILSM was replaced with *bDissolve*. The runs were started with random *k*-partitions. Quality in terms of percentage loss of information and execution elapsed time were recorded. The averages for the 20 runs were computed and used to compare ILSM with *bDissolve* to ILSM (with *dissolve*).

The *bDissolve* perturbation biases selection of clusters to dissolve toward clusters with higher information loss. To efficiently do this one or more clusters are selected at random and the one with the highest information loss is dissolved. The number of clusters selected per perturbation is called the sampling rate. Increasing the sampling rate shifts the bias from diversification toward intensification. If the sampling rate is one, then *bDissolve* and *dissolve* are equivalent, and diversification is at its maximum for *bDissolve*.

Preliminary experiments were run to determine the most overall effective sampling rate. Sampling rates greater than 10 had negative results. Sampling rates less than 4 did not provide enough biasing and the results were not significantly different compared to ILSM. The best sampling rate was found to be 5. The probability of selecting a cluster is discussed in Chapter 3.

Table 7 shows the quality resulting from ILSM with *bDissolve* compared to ILSM (with *dissolve*). At the end of 5000 iterations *bDissolve* produced slightly lower averages for information loss for 10 of the 15 benchmarks and was only slightly worse for one benchmark. Nonetheless, *bDissolve* had its greatest impact in the early iterations.

Figures 12 through 18 show how information loss decreases over successive iterations of ILSM with *bDissolve* compared to ILSM. The Figures show results over a representative set of test cases. For two datasets, *Tarragona* and *EIA*, *bDissolve* showed a clear advantage over *dissolve* at reducing information loss in the early iterations of ILSM, especially in the first 1000 iterations (see Figures 12 through 16). For these two datasets *bDissolve* consistently needed less than 500 iterations to achieved equivalent results to *dissolve* at 1000 iterations. As the number of iterations near 5000 the results for *bDissolve* and the original converge and *bDissolve* loses its advantage. Equivalent results were achieved for all values of *k* for these two datasets (*Tarragona and EIA*). The empirical results showed that there is much less opportunity to extract additional information loss beyond the extant best values.

| | k = 3 | k = 4 | k = 5 | k = 6 | k = 10 |
|---|---|---|---|---|---|
| *Tarragona* | | | | | |
| *bDissolve* Best | 14.49% | 17.10% | 20.17% | 23.50% | 30.14% |
| *bDissolve* Avg | 14.52% | 17.13% | 20.19% | 23.55% | 30.17% |
| *bDissolve* Worst | 14.58% | 17.16% | 20.22% | 23.62% | 30.27% |
| *dissolve* Best | 14.48% | 17.11% | 20.17% | 23.52% | 30.14% |
| *dissolve* Avg | 14.57% | 17.15% | 20.21% | 23.59% | 30.19% |
| *dissolve* Worst | 14.60% | 17.20% | 20.28% | 23.65% | 30.29% |
| *Census* | | | | | |
| *bDissolve* Best | 4.77% | 6.12% | 7.37% | 8.36% | 11.47% |
| *bDissolve* Avg | 4.80% | 6.16% | 7.41% | 8.41% | 11.53% |
| *bDissolve* Worst | 4.83% | 6.22% | 7.45% | 8.45% | 11.63% |
| *dissolve* Best | 4.77% | 6.13% | 7.38% | 8.33% | 11.47% |
| *dissolve* Avg | 4.80% | 6.19% | 7.44% | 8.41% | 11.54% |
| *dissolve* Worst | 4.83% | 6.30% | 7.53% | 8.50% | 11.69% |
| *EIA* | | | | | |
| *bDissolve* Best | 0.36% | 0.51% | 0.76% | 0.94% | 1.85% |
| *bDissolve* Avg | 0.37% | 0.52% | 0.78% | 0.96% | 1.86% |
| *bDissolve* Worst | 0.38% | 0.52% | 0.90% | 0.99% | 1.86% |
| *dissolve* Best | 0.37% | 0.52% | 0.76% | 0.94% | 1.85% |
| *dissolve* Avg | 0.38% | 0.52% | 0.82% | 0.95% | 1.86% |
| *dissolve* Worst | 0.41% | 0.54% | 0.96% | 0.97% | 1.86% |

**Table 7:** Information loss for runs using: *bDissolve* compared to *dissolve*

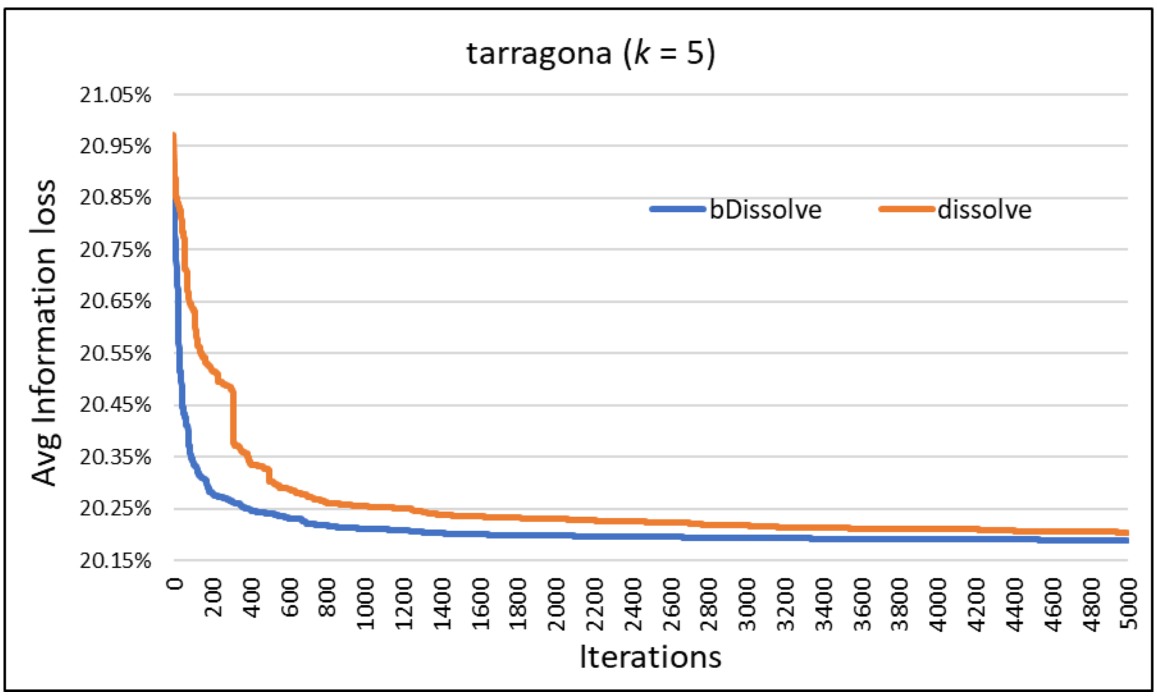


**Figure 12:** Decrease in information loss, *bDissolve* vs *dissolve* (*Tarragona* 5)

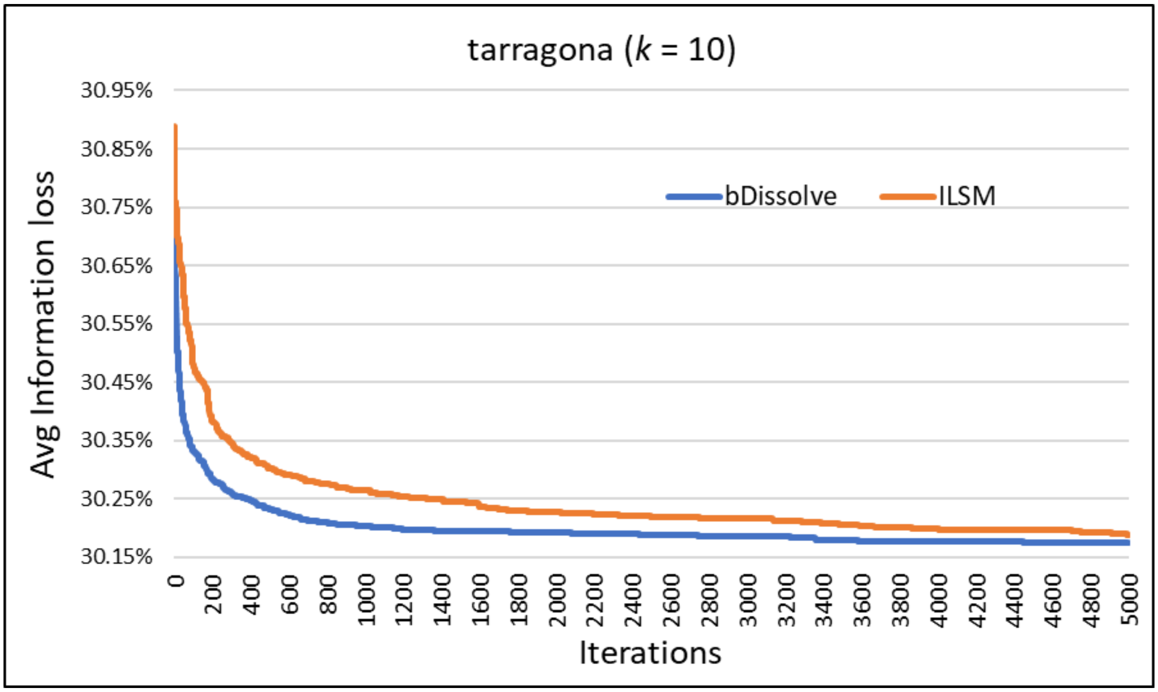


**Figure 13:** Decrease in information loss, *bDissolve* vs *dissolve* (*Tarragona* 10)

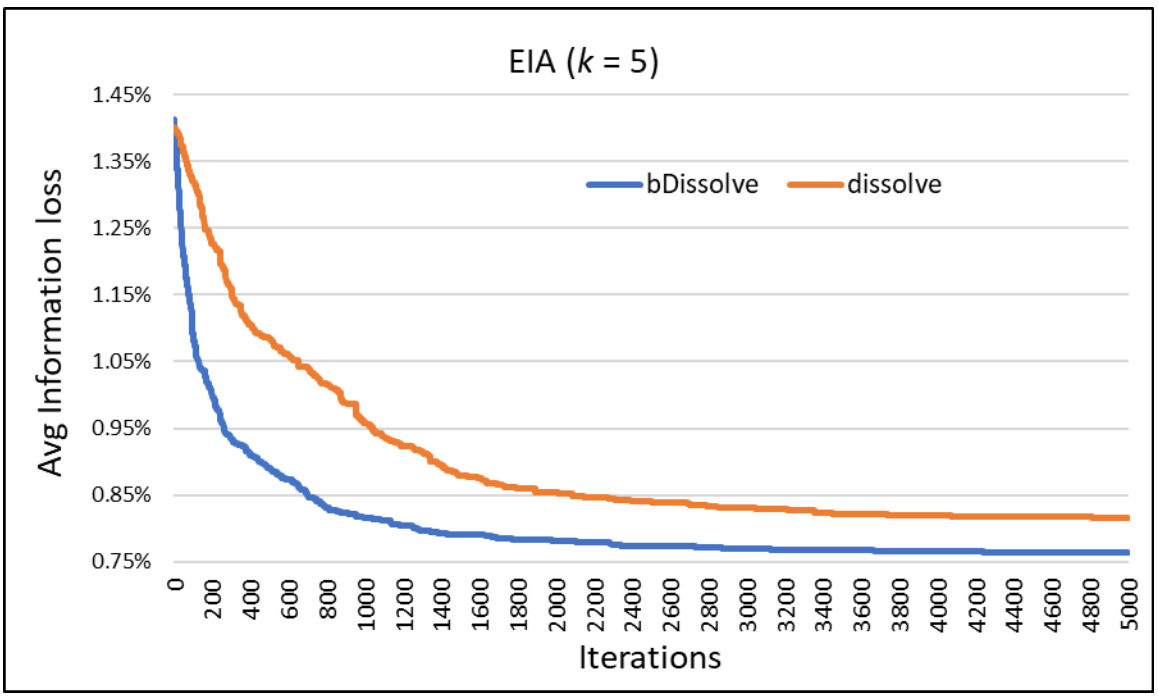


**Figure 14:** Decrease in information loss, *bDissolve* vs *dissolve* (*EIA* 5)

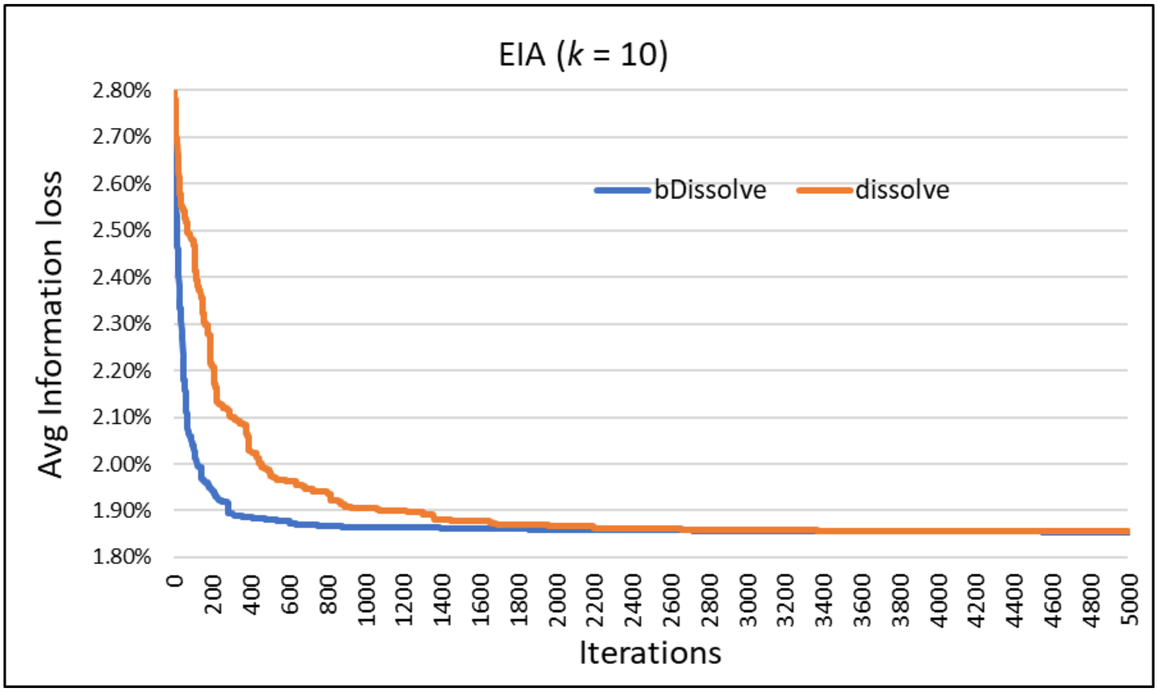


**Figure 15:** Decrease in information loss, *bDissolve* vs *dissolve* (*EIA* 10)

Figure 16 and 18 show that *bDissolve* is less effective for the *Census* dataset. As the value of *k* increases the effect decreases. For $k = 10$ no improvement was obtained. The data from the three datasets were compared but nothing stood out to explain why *Census* was less affected.

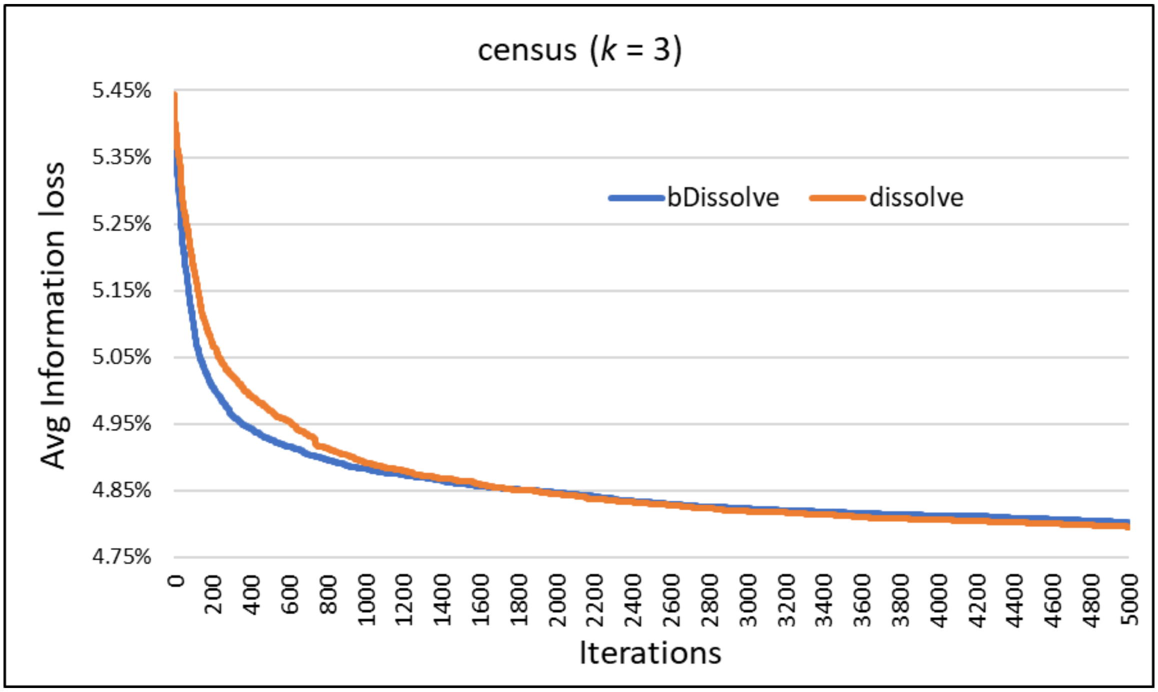


**Figure 16:** Decrease in information loss, *bDissolve* vs *dissolve* (*Census* 3)

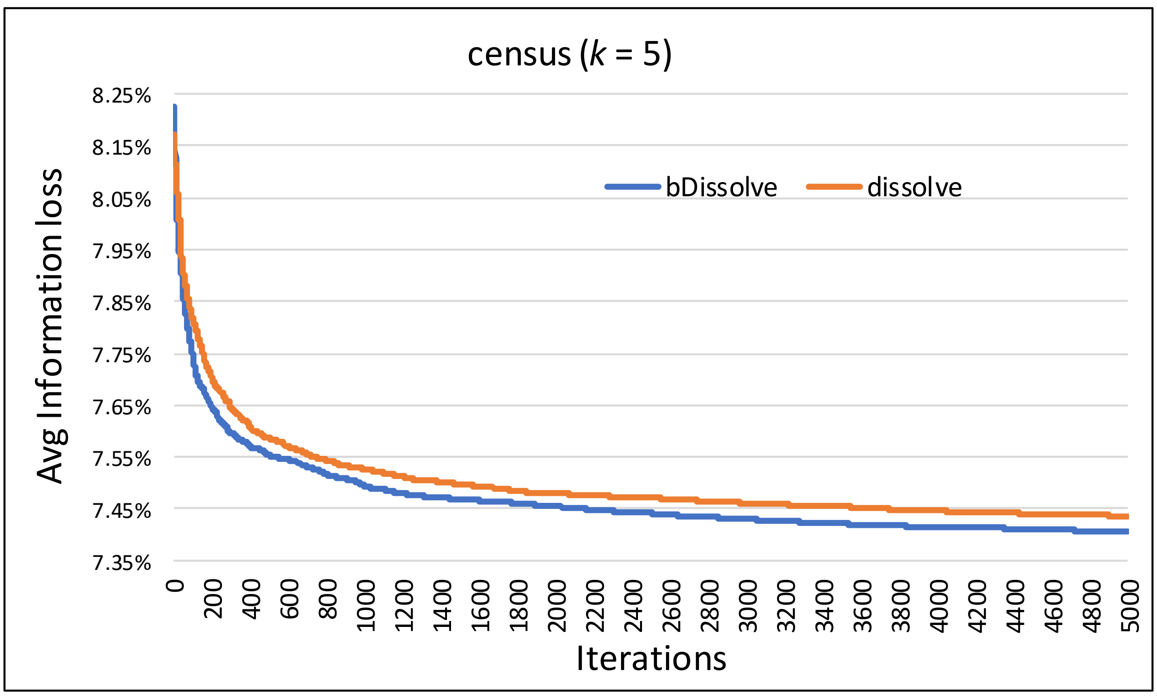


***Figure 17:** Decrease in information loss, bDissolve vs dissolve (Census 5)*

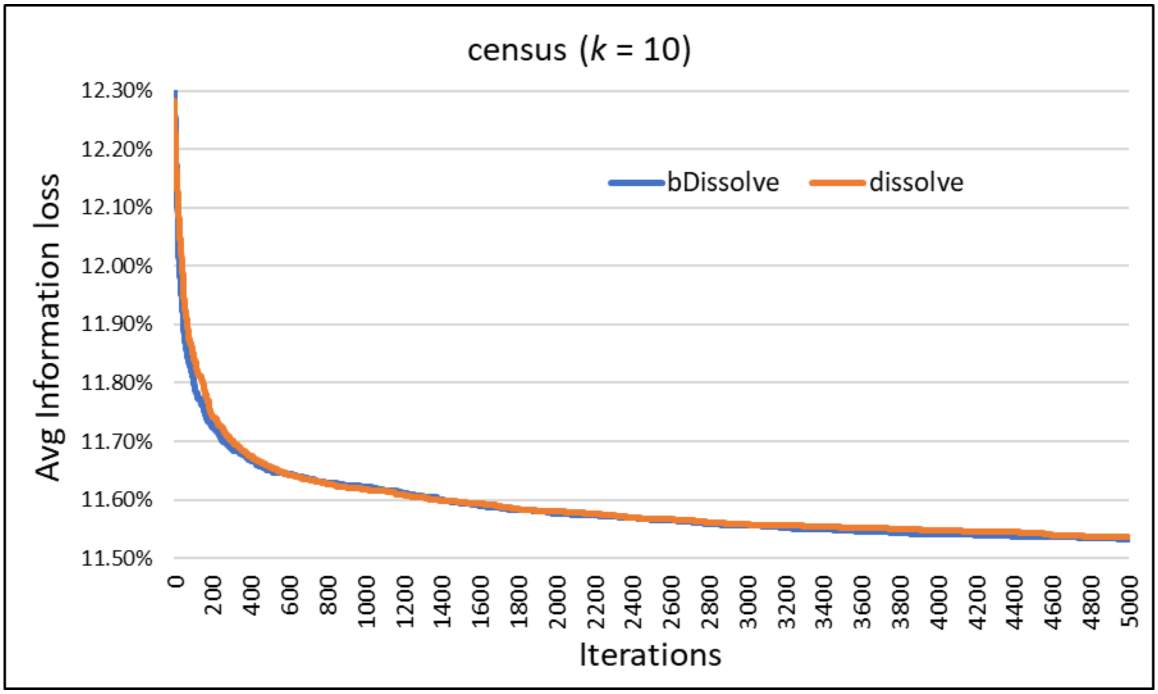


**Figure 18:** Decrease in information loss, *bDissolve* vs *dissolve* (*Census* 10)

**Dynamic Acceptance Criteria versus Static**

Experiments consisted of 20 runs of ILSM where each run terminated after 5000 iterations. The *acceptanceCriterion* method within ILSM was replaced with *dAcceptanceCriterion*. The runs were started with random *k*-partitions. Quality in terms of percentage loss of information and execution elapsed time were recorded. The averages for the 20 runs were computed and used to compare ILSM with *dAcceptanceCriterion* to ILSM (with *acceptanceCriterion*).

The *dAcceptanceCriterion* method dynamically changes the probability of acceptance based on the difference in quality between the current solution and the current best solution. Current solutions closer in quality to the current best solution are accepted with higher probability than ones with lesser quality. The method incorporates a tuning factor that must be determined for best performance. The original acceptance criterion had a static acceptance probability of 0.8 while *dAcceptanceCriterion* accepts solutions with the following probability where $T$ is the tuning factor.

$$P = e^{\left(\frac{(\mathrm{SSE}(bestP) - \mathrm{SSE}(P''))}{(T * \mathrm{SSE}(bestP))}\right)}, \text{ where } \mathrm{SSE}(bestP) < \mathrm{SSE}(P'')$$

A higher tuning factor increases the probability of acceptance; thus, increasing diversification. A lower tuning factor decreases the probability of acceptance; thus, increasing intensification. As with the original if the new solution has a lower sum of the squared error than the best known then it is accepted with a 100% probability. Preliminary experiments were run to determine the best overall tuning factor for *dAcceptanceCriterion*. The best tuning factor was found to be 0.00001.

Figure 19 through Figure 27 show how information loss decreases over successive iterations of ILSM with *dAcceptanceCriterion* compared to *acceptanceCriterion*. Positive effect from *dAcceptanceCriterion* diminished with increasing $k$, it provided slight or no improvement for all three datasets at $k = 10$. The dynamic acceptance criterion had the greatest effect for lower values of $k$ ($k = 3$, 4, 5 and 6), see Figure 19 through Figure 21. The exceptions were the mixed results obtained for *Tarragona*, $k = 5$ and $k = 6$ (Figure 24). It tended to help more in the early iterations. For *Tarragona*, $k = 10$ (Figure 27) it hurt the end results compared to ILSM. Overall dAcceptanceCriterion had the least impact of the three improvements.

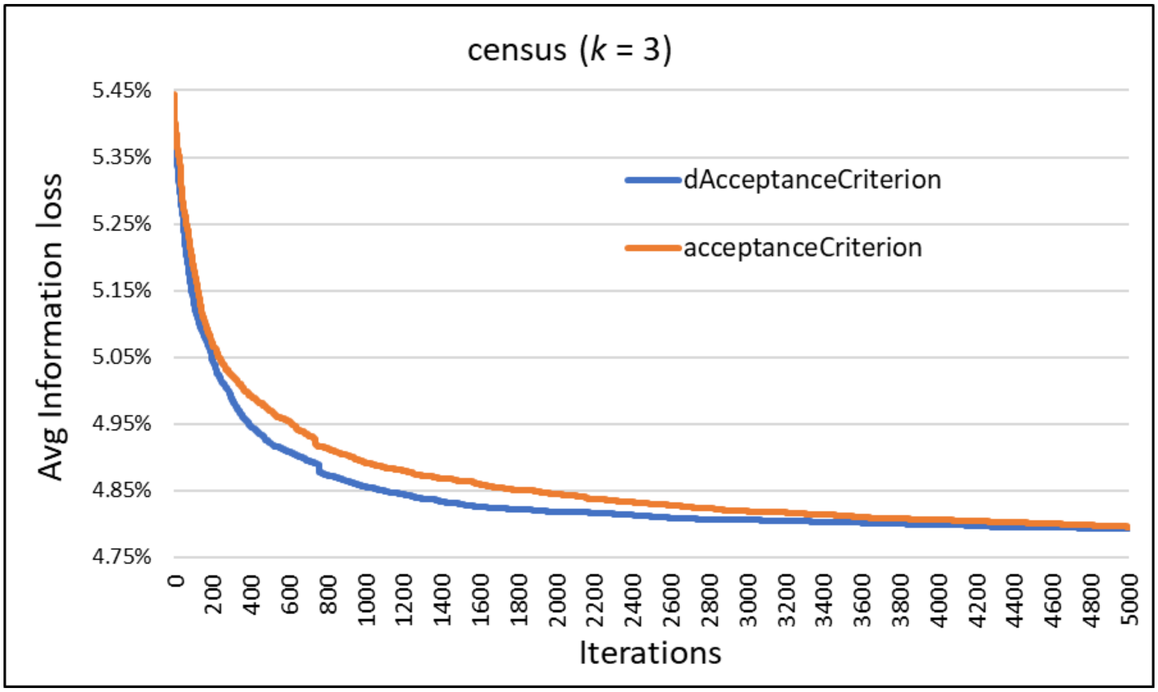


**Figure 19:** Decrease in information loss, *dAccept* vs *accept* (*Census* 3)

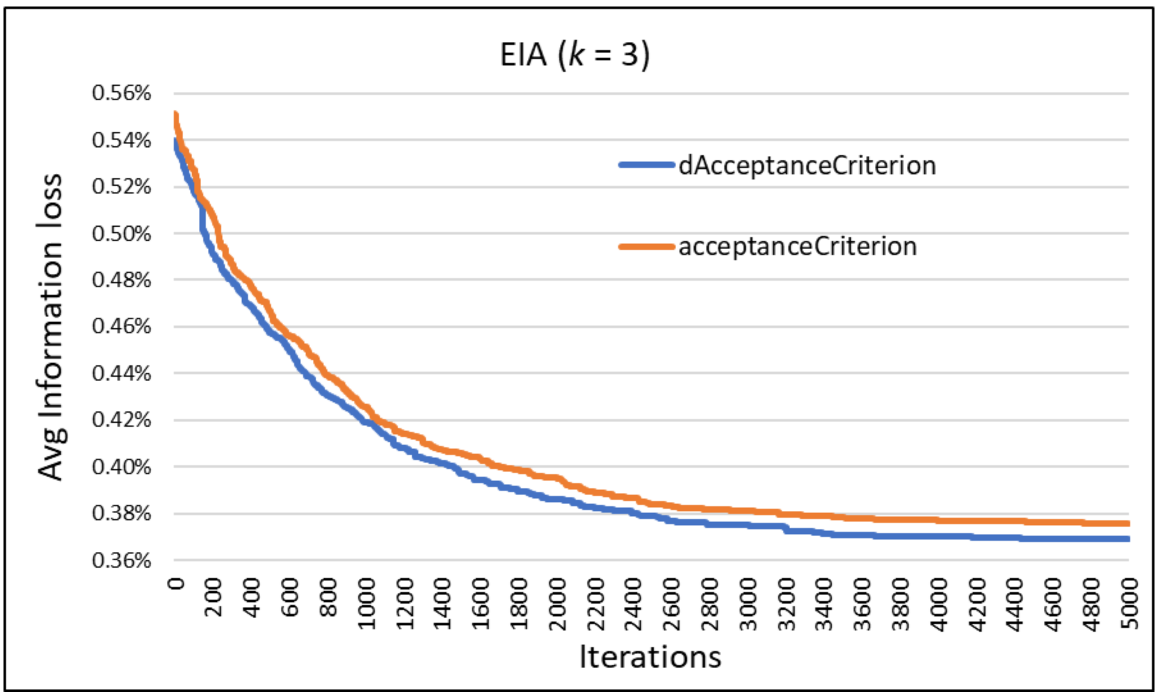


**Figure 20:** Decrease in information loss, *dAccept* vs *accept* (*EIA* 3)

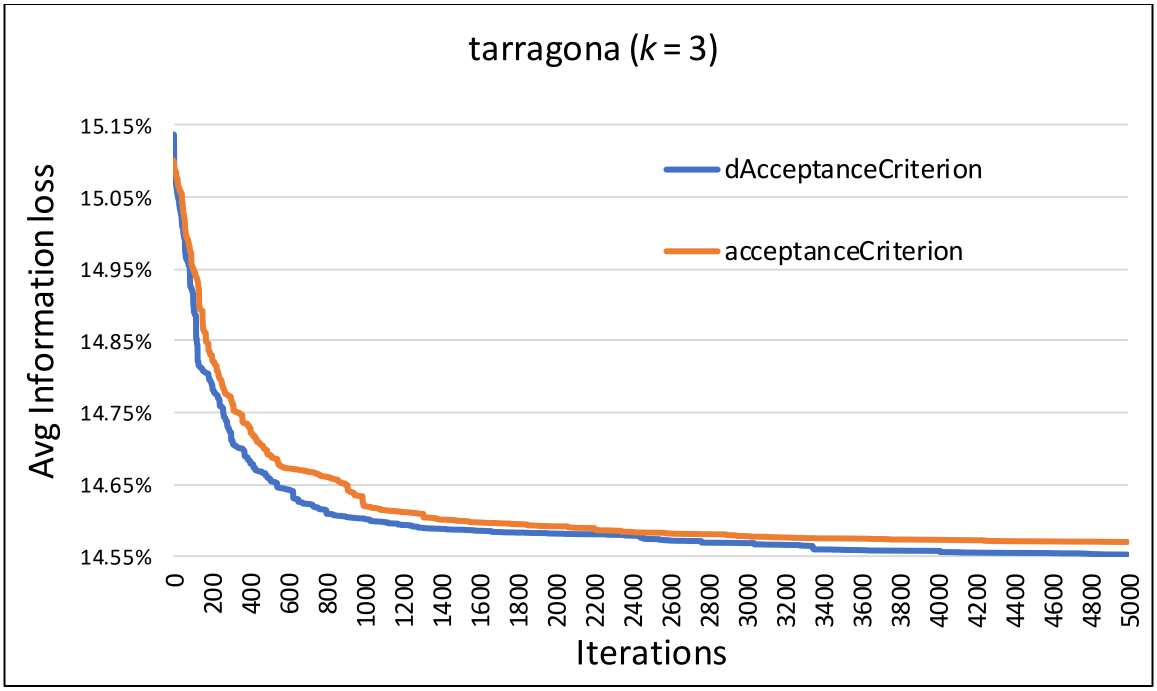


**Figure 21:** Decrease in information loss, *dAccept* vs *accept* (*Tarragona* 3)

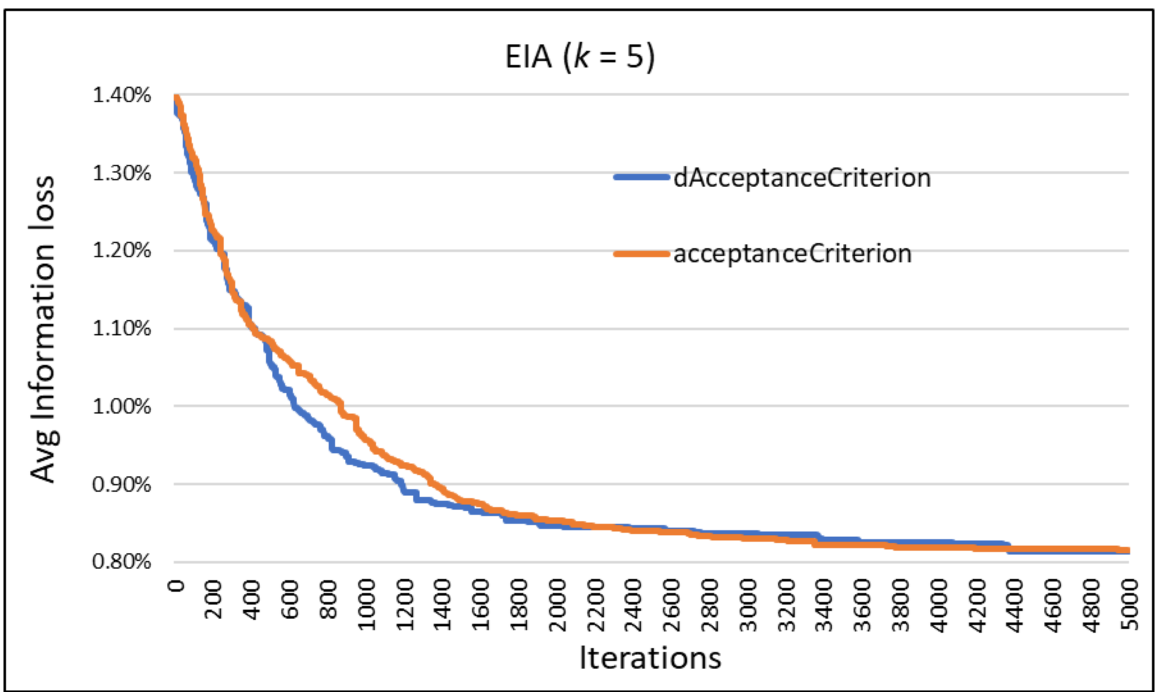


**Figure 22:** Decrease in information loss, *dAccept* vs *accept* (*EIA* 5)

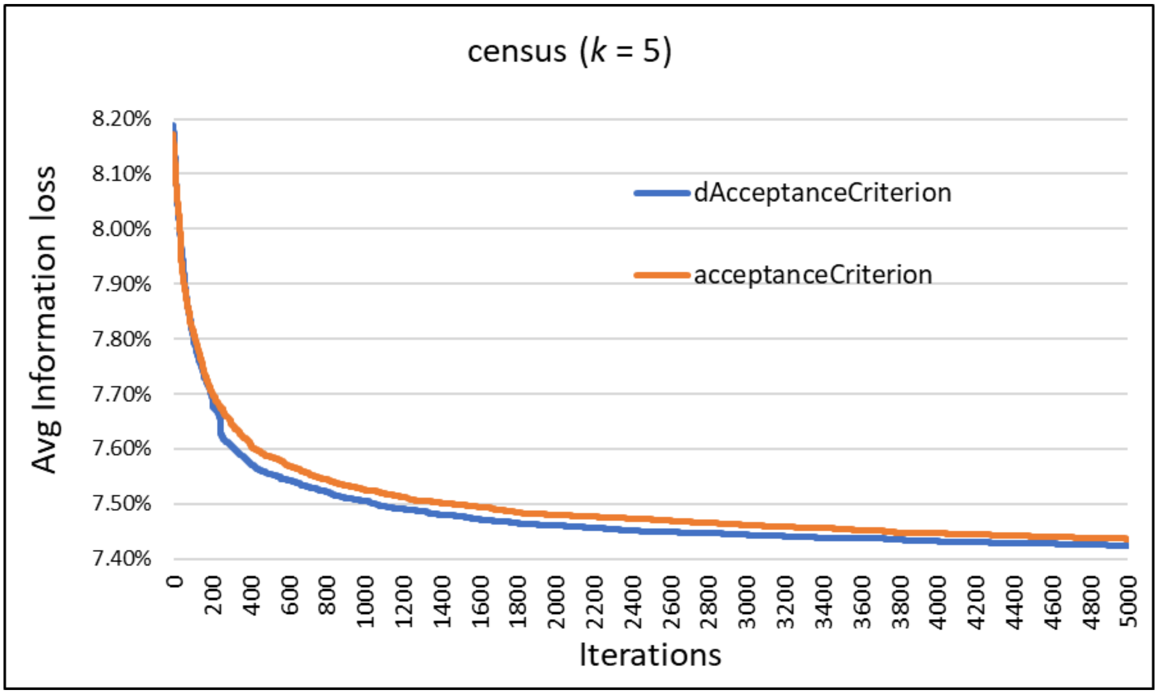


**Figure 23:** Decrease in information loss, *dAccept* vs *accept* (*Census* 5)

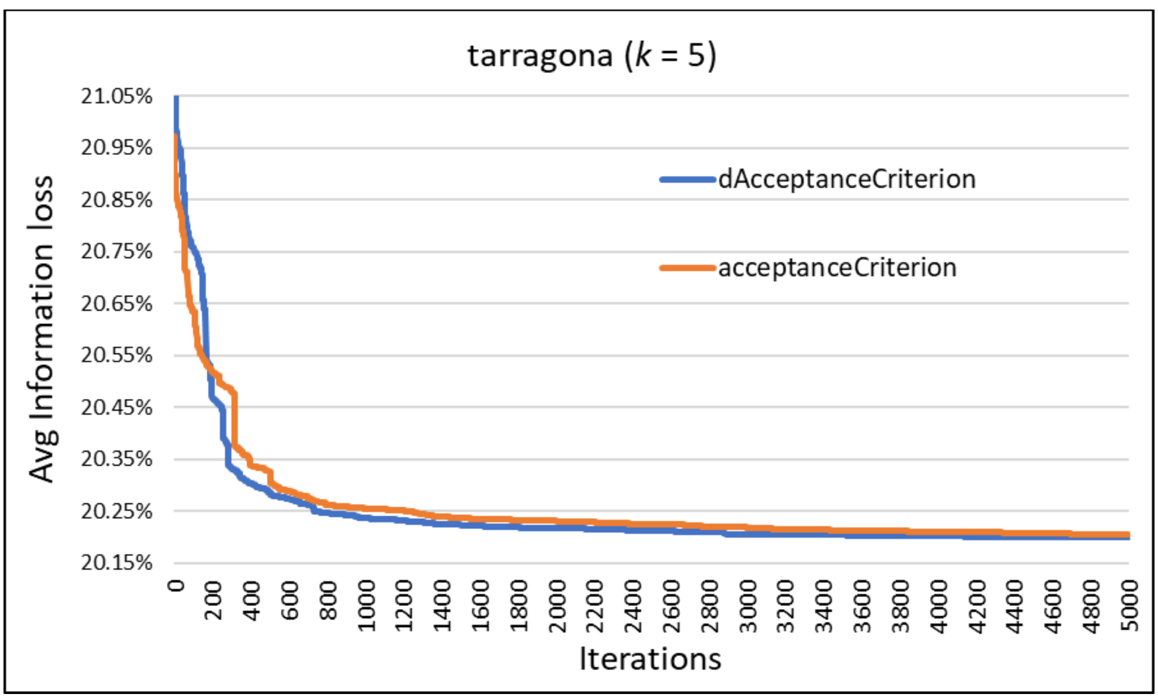


**Figure 24** Decrease in information loss, *dAccept* vs *accept* (*Tarragona* 5)

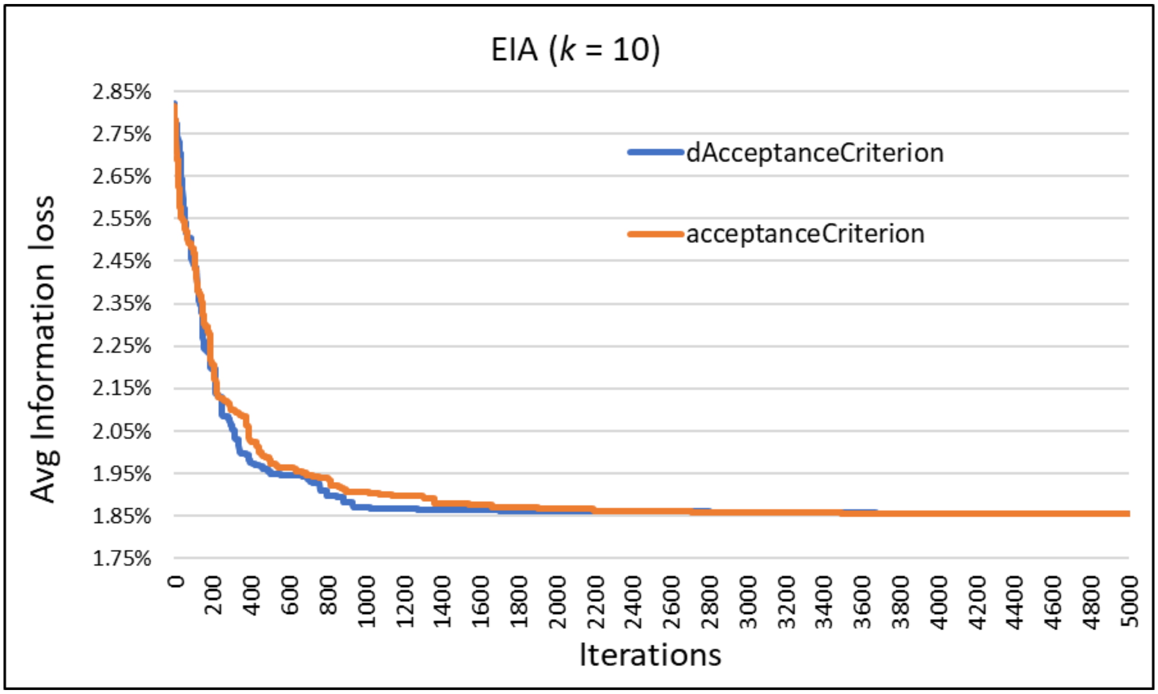


**Figure 25:** Decrease in information loss, *dAccept* vs *accept* (*EIA* 10)

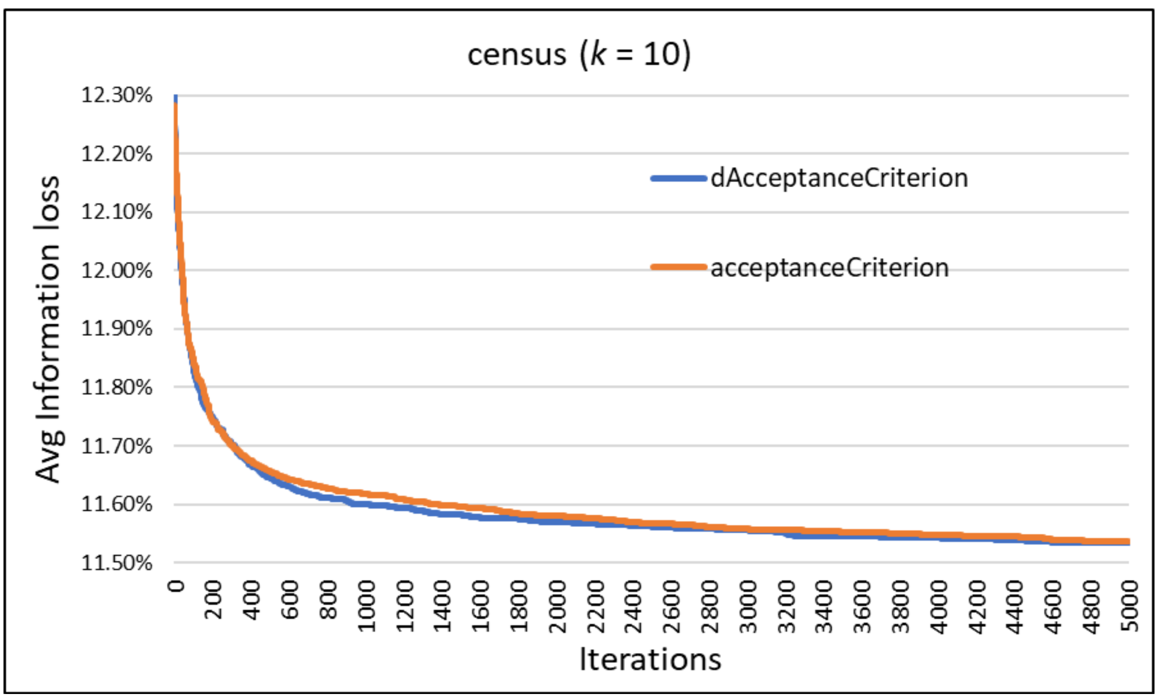


**Figure 26:** Decrease in information loss, *dAccept* vs *accept* (*Census* 10)

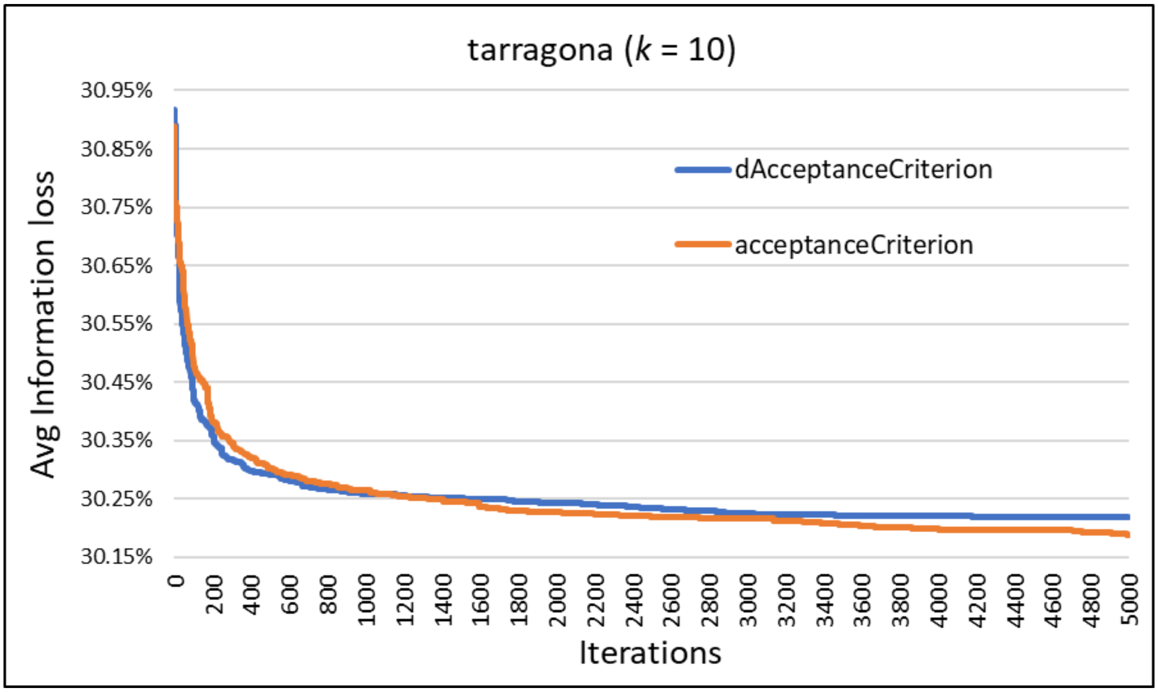


**Figure 27:** Decrease in information loss, *dAccept* vs *accept* (*Tarragona* 10)

### Advantages of Combining *bDissolve* and *dAcceptanceCriteria*

Multiple runs where used to study the advantages of using both *bDissolve* and *dAcceptanceCriteria* in the iterated local search. Experiments consisted of 20 runs each of the iterated local search. The runs were started with random *k*-partitions. Each run of ILS performed 5000 iterations before the terminal condition was met. The results were then averaged for comparison purposes.

Both *bDissolve* and *dAcceptanceCriteria* complemented each other and their advantages were demonstrated to be additive in some test cases. Both methods operate in two quite different areas of the iterated local search. They are designed to operate exclusive of each other and there is no obvious overlap. The results did not show either one handicapping the other.

Figure 28 through Figure 33 show how information loss decreases over successive iterations of ILSM using both *bDissolve* and *dAcceptanceCriterion* compared to ILSM and when *bDissolve* and *dAcceptanceCriterion* were used individually. Results for *Census* and *EIA* (*k* = 3, 4,5 and 6) were improved upon by combining *bDissolve* and *dAcceptanceCriterion* compared to ILSM (see Figure 28 - Figure 31, *k* = 4 and *k* = 6 not shown). As before the most impact was seen in the earlier iterations. Very little improvement was obtained by the *Census* and *EIA* experiments for *k* = 10, see Figure 32 and Figure 33. The *Tarragona* dataset for all values of *k* saw no advantage to combining the features as compared to ILSM with *bDissolve* by itself, Figure 34 is representative of all the results for the *Tarragona* dataset.

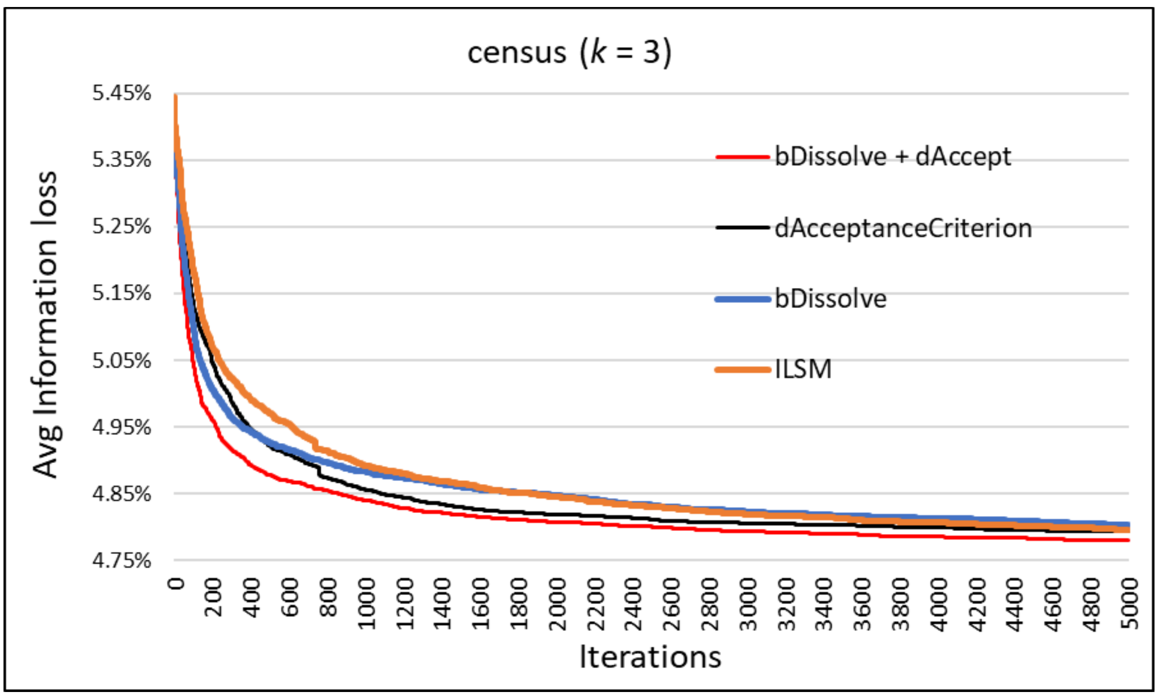


**Figure 28:** Decrease in information loss, *bDissolve + dAccept* vs ILSM (*Census* 3)

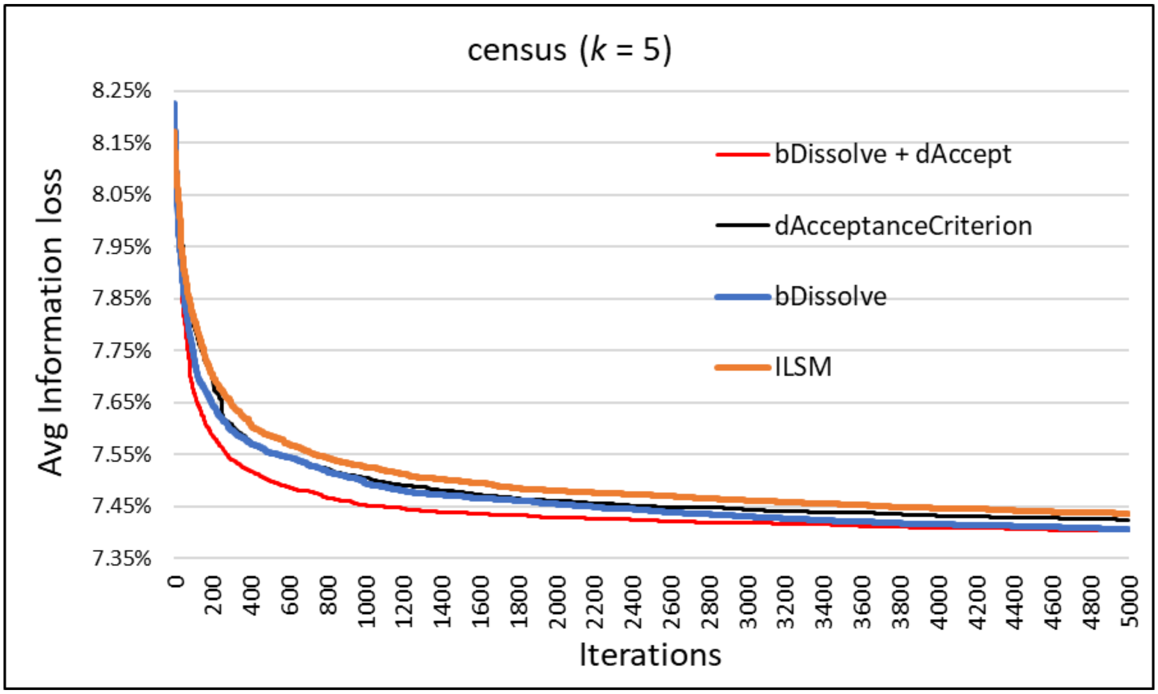


**Figure 29:** Decrease in information loss, *bDissolve + dAccept* vs ILSM (*Census* 5)

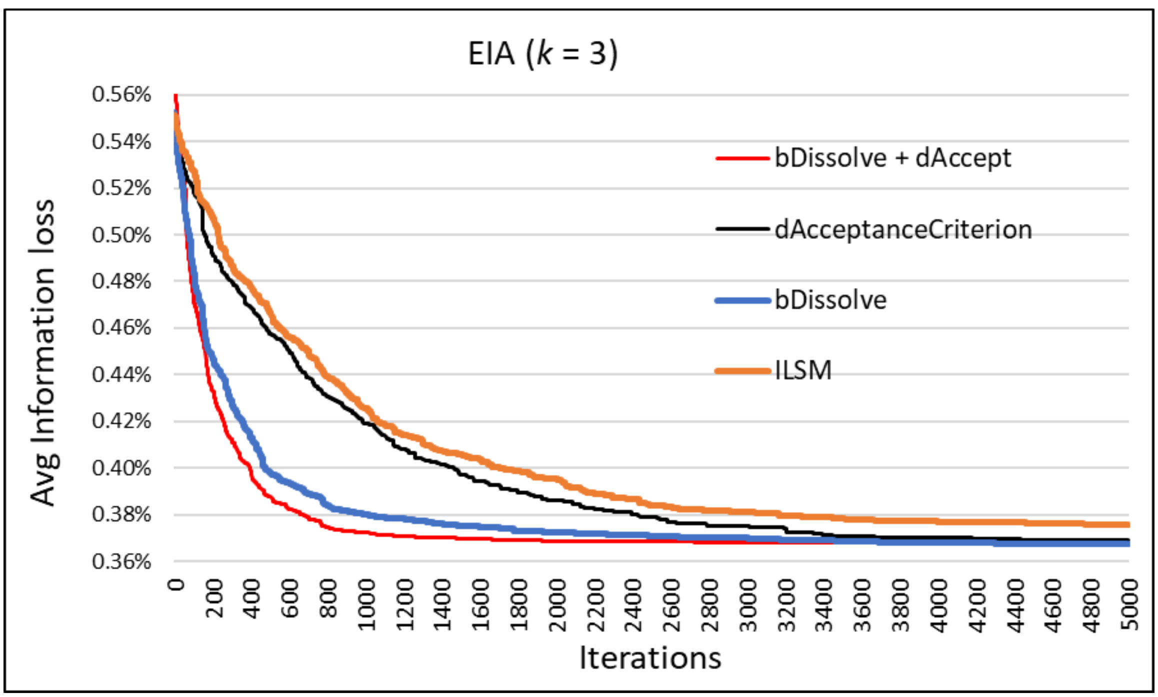


**Figure 30:** Decrease in information loss, *bDissolve + dAccept* vs ILSM (*EIA* 3)

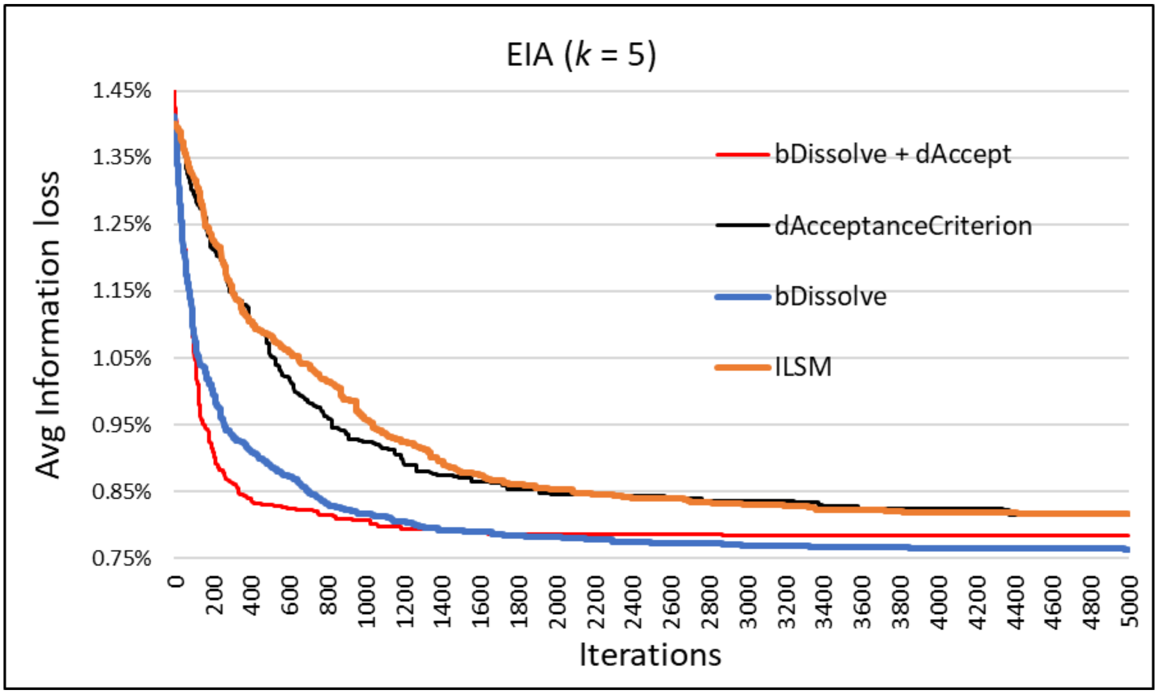


**Figure 31:** Decrease in information loss, *bDissolve + dAccept* vs ILSM (*EIA* 5)

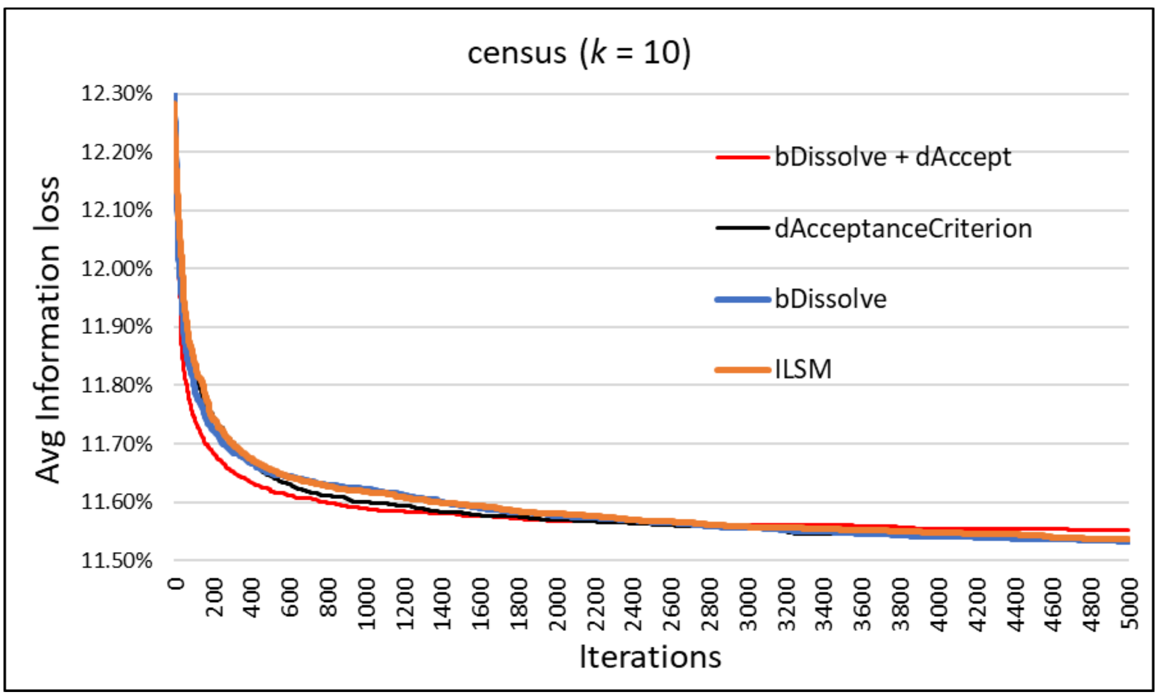


**Figure 32** Decrease in information loss, *bDissolve + dAccept* vs ILSM (*Census* 10)

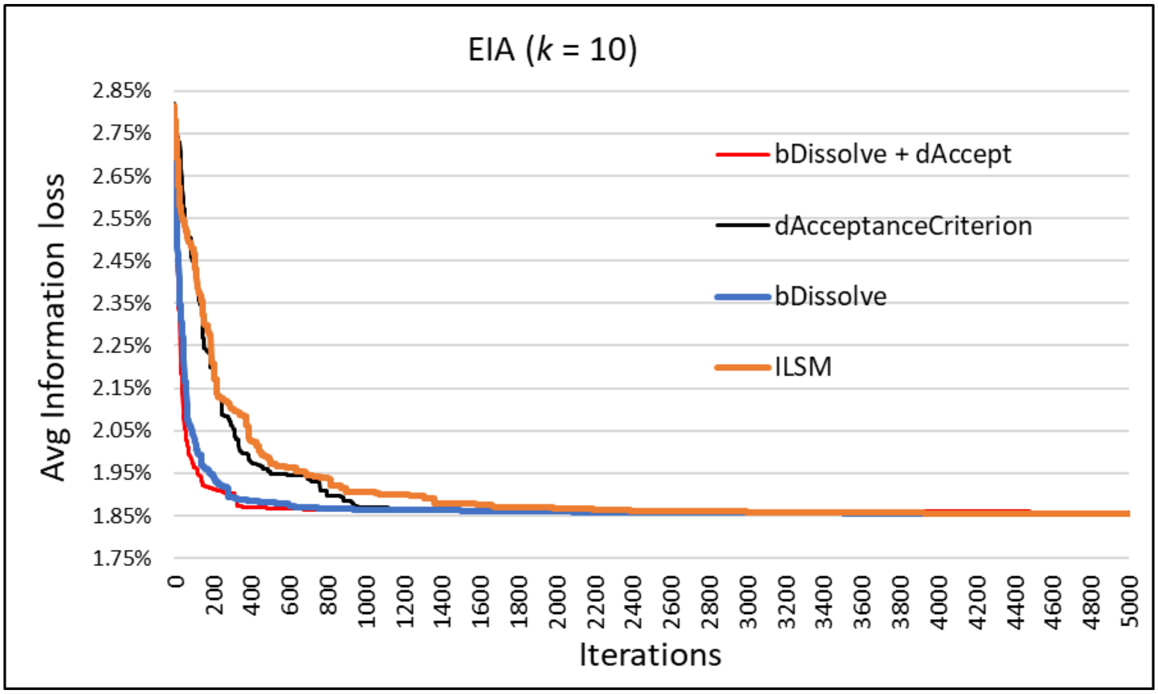


**Figure 33:** Decrease in information loss, *bDissolve + dAccept* vs ILSM (*EIA* 10)

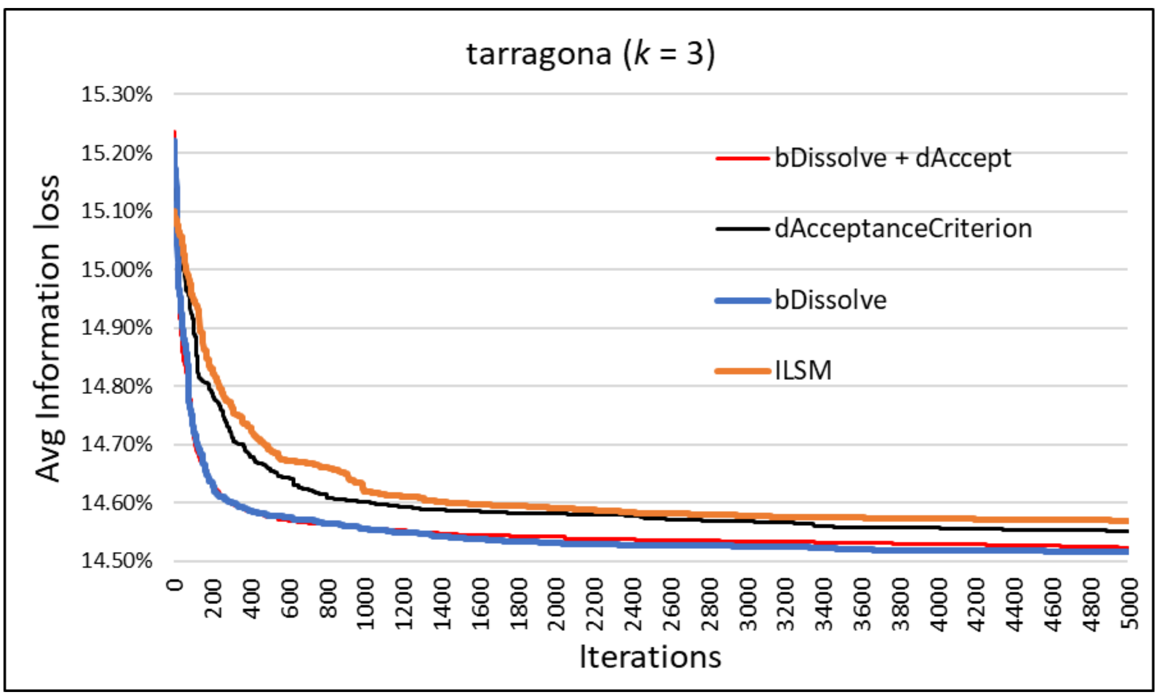


**Figure 34:** Decrease in information loss, *bDissolve + dAccept* vs ILSM (*Tarragona* 3)

# Chapter 5

## Conclusions, Implications, Recommendations, and Summary

This chapter draws conclusions and details implications from the observations, results and findings gathered in this study. From these it makes recommendations for future study and concludes with an overall summary of the study.

Chapter 1 presented the problem statement concerning performance improvements of Iterated Local Search for Microaggregation (ILSM) (Laszlo & Mukherjee, 2015). It did so in terms of quality (percentage information loss) and speed (elapsed time). Three research questions were posed within the problem statement. The first question was based on recommendations for further study in Laszlo and Mukherjee's original paper (2015). The second and third questions were based on suggestions in previous research in metaheuristics (Blum & Roli, 2003) detailed in the literature review. The first question, RQ1, asked whether the costliest part of LS, tests of whether a pair of clusters can swap or shift points, could be avoided. RQ2 asked whether better cluster candidates to perturb could be efficiently selected to speed the reduction of latent information loss. RQ3 asked whether dynamically changing the acceptance criterion would select better solutions that escape local minimums faster.

As a response to these questions, Chapter 3 proposed three novel methods which led to improvements. The first added cluster change tracking to ILSM creating ILSMC. It added cluster change tracking to LS, ILSM and the perturbation operations. The second was the biased *dissolve* perturbation, *bDissolve*, which biased the perturbation operator toward clusters with higher loss. The third was the dynamic acceptance criterion, *dAcceptanceCriterion*. It dynamically changed the probability of acceptance based on the difference in quality between the current solution and the best solution. Current solutions closer in quality to the best solution were accepted with higher probability than ones with lesser quality.

**Conclusions**

Chapter 4 presented the results from the experiments conducted in this study. Quality and speed were recorded consistent with the methodology in Chapter 3. ILSMC had the greatest impact over ILSM. It significantly reduced the elapsed times of iterations while producing solutions of similar quality. The new perturbation operation *bDissolve* significantly increased the rate of quality improvement for the first 250 to 500 iterations over the original *dissolve*. The dynamic acceptance criterion *dAcceptanceCriterion* had less impact than the *bDissolve* operation, nonetheless, it also increased the rate of quality improvement for the first 250 to 500 iterations over the original static acceptance criterion. When all three were combined, improvements were orthogonal and additive, and the overall best results were realized compared to ILSM.

Both *bDissolve* and *dAcceptanceCriterion* had lesser effect in the latter iterations as resulting solutions neared the extant best values of the original ILSM. The difference in quality between *bDissolve* and *dissolve* was small over the last 2000 iterations. The *bDissolve* operation needed 2000 less iterations for most test cases to consistently reach extant best quality compared to *dissolve*; however, it had the least effect on the *Census* dataset.

The *dAcceptanceCriterion* had the least impact of all the improvements. Better results compared to ILSM tended to be in the earlier iterations but tended to produce similar quality compared to *acceptanceCriterion* after that point. Using both improvements together was better than either by themselves for the *Census* and *EIA* datasets, but not the *Tarragona* dataset. The dynamic acceptance criterion is clearly sensitive to the type of data or the best tuning factor was not selected. For some test cases the extant best solutions were consistently matched in 3000 to 4000 iterations as compared to ILSM which consistently needed 4000 to 5000 iterations.

RQ1 concerns the effectiveness of tracking cluster changes to avoid tests to significantly reduce execution run times. LSC avoided testing many cluster pairs when compared to the original LS where all cluster pairs are tested (i.e. *maySwap*, *mayShift*, *swapTest* and *shiftTest*). When LSC was run in a random restart regime, the average execution times were reduced by 30% to 50% compared to LS (see Table 1).

Early in the study it became obvious many more swap and shift tests could be avoided within ILSMC by also tracking the clusters changed within the perturbation operations not just LSC. With cluster change tracking utilized in both LSC and the perturbation operations, average execution times for ILSMC were reduced by 78% to 99.5% compared to ILSM (see Table 2).

The research question also asked whether quality would be affected. The most impactful changes by far were the avoided shift and swap tests. But avoiding unneeded tests has no bearing on next steps along the local search path, only execution speed. The average quality at any iteration over the 5000 iterations was not expected to be impacted for the better. Only the time for each iteration was expected to be reduced. The results confirmed the average quality at any iteration count remained the same for ILSM and ILSMC. ILSMC just executed each iteration much faster. However, a concern did lie with partially lifting of the requirement to randomize the order of testing the cluster pairs. The concern was that reducing randomness would negatively impact quality, despite this the *updateC* procedure proved resilient and the quality of the results remained at extant best levels.

The results also provided considerable empirical evidence that ILSMC reduces computational complexity compared to ILSM. While ILSM was dominated by $O(n^2)$ complexity within the *update* procedure of LS, ILSMC was dominated by $O(n)$ complexity within the *updateC* procedure of LSC reducing execution times by 70% to 99.5%. Only on the first call to LSC was ILSMC dominated by $O(n^2)$ complexity.

To improve quality, RQ2 concerns the effectiveness of biasing the selection of clusters to perturb toward ones with higher loss. The perturbation operations in ILSM selected clusters at random. Selecting clusters to perturb with higher loss proved effective in the early iterations of the experiment runs. The average number of iterations needed to produce comparable results to the original were reduced by approximately 20% to 50%. However, the results suggested that there is limit to the amount of latent information loss that can be removed and the current best extant values are near that limit. Occasionally new best values were found, they were only slightly better than those in the original Laszlo and Mukherjee paper (2015). The most dramatic improvements for *bDissolve* were in the first 250 to 500 iterations where the latent information loss at a certain iteration counts were reduced by approximately 15% to 60% over *dissolve*.

RQ3 considered the effectiveness of a dynamic acceptance criterion at improving quality over the original static criterion (a fixed percentage.) This method was like the previous method in that it tries to improve search efficiency as oppose to computational efficiency. The intent was to dynamically vary the probability of accepting a solution based on the difference in quality between the current solution and the current best solution. Current solutions closer in quality to the current best solution were accepted with higher probability than ones with lesser quality. The *dAcceptanceCriterion* was only effective for values $k = 6$ or less. It was ineffective for the *Tarragona* dataset which could have been the wrong tuning factor was chosen or that *dAcceptanceCriterion* is sensitive to something in the makeup of the data. Like *bDissolve*, it tended to be effective in the early iterations of the test runs. However, the quality of the results after 5000 iterations were practically the same reinforcing the notion that most of the latent information loss is already removed by the end of the run. As with *bDissolve*, the most improvement for *dAcceptanceCriterion* were in the earlier iterations.

## Implications

This study has shown that tracking cluster changes in ILSMC avoids most of the shift and swap tests reducing elapsed execution times over the original ILSM. Also, ILSM runs with $O(n^2)$ complexity, while ILSMC runs with $O(n)$ complexity in practice for all but the first call to LSC which runs with $O(n^2)$ complexity. This is important because this allows ILSMC to scale with larger datasets better than ILSM.

The biased perturbation operation and the dynamic acceptance criterion allows ILSM and ILSMC to operate with fewer iterations. They are more effective when there are substantial amounts of latent information loss. It is also sensitive to the makeup of the data and is not effective with all datasets and values of *k*. Also, the ability to quickly improve quality might be useful to interactive and online applications. Also, they may prove more effective when latent information loss is not easily recovered with simple perturbation and acceptance operations.

## Recommendations

During this study it was observed that perturbations and subsequent swaps and shifts were very localized. After the first call to LS and LSC each iteration touched a very localized portion of the partition, see Figure 9. The broader question is whether methods can be developed within the framework of iterated local search to take advantage of this localization. A further line of research could address how a partition may be divided into isolated locales to allow parallel perturbations operations and parallel solutions in general. Parallel methods could further increase speed and scalability of ILSMC to the benefit of microaggregation and *k*-anonymization users with the largest datasets.

## Summary

Microaggregation for producing *k*-anonymity is widely employed to protect microdata from disclosure. Laszlo and Mukherjee (2015) presented a microaggregation method, Iterated Local Search for Microaggregation (ILSM). It consistently identifies better quality solutions on instances of benchmark problems than all other extant heuristics. However, speed is a practical problem and ILSM does not scale well as the size of the dataset increases. Slow processing generally limits its use to offline processing (Laszlo & Mukherjee, 2015). This study demonstrates that ILSM can be significantly improved in three orthogonal and additive ways with ILSMC, *bDissolve*, and *dAcceptanceCriterion*. ILSMC will be especially useful to data scientist and data owners where datasets are too large for practical solution by ILSM.

# References


Adam, N. R., & Worthmann, J. C. (1989). Security-control methods for statistical databases: a comparative study. *ACM Computing Surveys*, *21*(4), 515–556. https://doi.org/10.1145/76894.76895

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: overview and conceptual comparison. *ACM Computing Surveys*, *35*, 189–213. https://doi.org/10.1007/s10479-005-3971-7

Brucker, P. (1978). On the Complexity of Clustering Problems. In *Optimization and operations research* (pp. 45–54). Springer Berlin Heidleberg.

Campan, A., & Truta, T. M. (2009). Data and structural *K*-anonymity in social networks. *Privacy, Security, and Trust in KDD*, 33–54. https://doi.org/10.1007/978-3-642-01718-6_4

Chang, C., Li, Y., & Huang, W. (2007). TFRP: An efficient microaggregation algorithm for statistical disclosure control. *Journal of Systems and Software*. Retrieved from http://www.sciencedirect.com/science/article/pii/S0164121207000611

Domingo-Ferrer, J., Martinez-Balleste, A., Mateo-Sanz, J., & Sebe, F. (2006). Efficient multivariate data-oriented microaggregation. *VLDB Journal*, *15*(4), 355–369. https://doi.org/10.1007/s00778-006-0007-0

Domingo-Ferrer, J., & Mateo-Sanz, J. (2002). Practical Data-oriented Microaggregation for Statistical Disclosure Control. *IEEE Transactions on Knowledge and Data Engineering*, *14*(1), 189–201.

Domingo-Ferrer, J., Sebe, F., & Solanas, A. (2008). A polynomial-time approximation to optimal multivariate microaggregation. *Computers and Mathematics with Applications*, *55*(4), 714–732. https://doi.org/10.1016/j.camwa.2007.04.034

Domingo-Ferrer, J., & Torra, V. (2005). Ordinal, continuous and heterogeneous *k*-anonymity through microaggregation. *Data Mining and Knowledge Discovery*, *11*(2), 195–212. https://doi.org/10.1007/s10618-005-0007-5

Duncan, G. T., Elliot, M., & Salazar-González, J.-J. (2011). *Statistical Confidentiality: Principles and Practice*. New York, NY: Springer New York.

Fienberg, S. E. (2005). Confidentiality and disclosure limitation. *Encyclopedia of Social Measurement*, *1*, 463–469. https://doi.org/http://dx.doi.org/10.1016/B0-12-369398-5/00053-0

Goldberger, J., & Tassa, T. (2010). Efficient anonymizations with enhanced utility. *Transactions on Data Privacy*, *3*(2), 149–175. https://doi.org/10.1109/ICDMW.2009.15

Hansen, S. L., & Mukherjee, S. (2003). A polynomial algorithm for optimal univariate microaggregation. *IEEE Transactions on Knowledge and Data Engineering*, *15*(4), 1043–1044. https://doi.org/10.1109/TKDE.2003.1209020

Health Insurance Portability and Accountability Act of 1996. (1996), 98.

Kokolakis, G., & Fouskakis, D. (2009). Importance partitioning in micro-aggregation. *Computational Statistics and Data Analysis*, *53*(7), 2439–2445. https://doi.org/10.1016/j.csda.2008.09.028

Laszlo, M., & Mukherjee, S. (2007). A genetic algorithm that exchanges neighboring centers for *k*-means clustering. *Pattern Recognition Letters*, *28*(16), 2359–2366. https://doi.org/10.1016/j.patrec.2007.08.006

Laszlo, M., & Mukherjee, S. (2015). Iterated local search for microaggregation. *Journal of Systems and Software*, *100*, 15–26. https://doi.org/10.1016/j.jss.2014.10.012

Mateo-Sanz, J., & Domingo-Ferrer, J. (1998). A comparative study of microaggregation methods. *Questiió*, *22*(3), 511–526.

National Institute of Standards and Technology. (2014). Security and Privacy Controls for Federal Information Systems and Organizations. https://doi.org/10.6028/NIST.SP.800-53Ar4

Oganian, A., & Domingo-ferrer, J. (2001). On the complexity of optimal microaggregation for statistical disclosure control. *Statistical Journal of the United Nations Economic Commission for Europe*, *4*, 345-353.

Oganian, A., & Domingo-Ferrer, J. (2001). On the complexity of optimal microaggregation for statistical disclosure control. *Statistical Journal of the United*. Retrieved from http://content.iospress.com/articles/statistical-journal-of-the-united-nations-economic-commission-for-europe/sju00495

Oommen, B. J., & Fayyoumi, E. (2010). On utilizing association and interaction concepts for enhancing microaggregation in secure statistical databases. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, *40*(1), 198–207. https://doi.org/10.1109/TSMCB.2009.2024949

Panagiotakis, C., & Tziritas, G. (2011). Successive Group Selection for. *IEEE Transactions on Knowledge and Data Engineering*, 1–6. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6095550

Panagiotakis, C., & Tziritas, G. (2013). Successive group selection for microaggregation. *IEEE Transactions on Knowledge and Data Engineering*, *25*(5), 1191–1195. https://doi.org/10.1109/TKDE.2011.242

Rebollo-Monedero, D., Forné, J., & Soriano, M. (2011). An algorithm for *k*-anonymous microaggregation and clustering inspired by the design of distortion-optimized quantizers. *Data and Knowledge Engineering*, *70*(10), 892–921. https://doi.org/10.1016/j.datak.2011.06.005

Samarati, P. (2001). Protecting respondents identities in microdata release. *And Data Engineering, IEEE Transactions On*. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=971193

Sweeney, L. (2002). *k*-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowlege-Based Systems*, *10*(5), 557–570. https://doi.org/10.1142/S0218488502001648

United Nations General Assembly. (2014). *Fundamental Principles of Official Statistics* (No. A/RES/68/261). Retrieved from http://undocs.org/A/RES/68/261