An Enhanced Guided Local Search for the QAP

Brubaker, Allen

Computer Science, Pennsylvania State University

Harrisburg, PA

ajb5377@psu.edu

*Abstract*—The Quadratic Assignment Problem (QAP) is one of the most complex NP-Hard combinatorial optimization problems remaining intractable for n>30. However, due to its applicability to many important scientific domains, many meta-heuristic approaches have been successfully implemented. The Guided Local Search (GLS) is one such meta-heuristic that guides a local search over a modified solution landscape induced by an augmented objective function. This paper details a series of enhancements to the basic GLS and is shown to be competitive if not better than GLS.

Keywords—quadratic assignment problem, guided local search

# Introduction

The Quadratic Assignment Problem (QAP), due to its importance both theoretically and practically stemming from many vital applications to a broad range of fields, has garnered increased research interest in recent years. It is a problem that entails assigning a set of facilities to another set of locations in an optimal manner [1]. Since there exists an inherent distance between each location and a flow demand between each facility, assigning one facility to a particular location impacts the entire system as this facility’s various flows must propagate varying distances to the other facilities that occupy corresponding locations. To be precise, the goal is to find an optimal permutation that induces an assignment of facilities to locations such that the sum of products between all flows and corresponding distances is minimal. Koopmans and Beckmann first introduced the problem in 1957 for several economic activities in the formal mathematical definition in what is known as the Koopmans-Beckman QAP [2]. Given two matrices and find the following permutation .

(1)

The QAP, since its origins in 1957, has been practically applied in many scientific domains. These have included “minimizing the number of connections in backboard wiring, economics, assignment of new facilities, scheduling problems, archeology, statistical analysis, analysis of reaction chemistry, numerical analysis, error control in communications, and memory layout optimization in signal processors” [2]. However, it has attracted the most attention in facility-layout problems [2]. These have included “assignment of buildings in a university campus, design of typewriter keyboards and control panels, hospital planning, forest management, facilities layout for minimizing work-in-process, and placement of electronic components” [2].

Many famous combinatorial optimization problems can be seen as special cases of QAP, such as Traveling Salesman, Maximum Clique, Graph Partitioning, and the Band-Width Reduction problems [1]. QAP is considered one of the most demanding combinatorial optimization problem not only because it has been proven NP-hard, but even finding an -approximation algorithm remains formidably NP-hard. Moreover, many exact methods show very poor performance and remain intractable for problem sizes beyond 30. Thus iterative-improvement heuristics to sophisticated meta-heuristics have been applied with drastic results [3].

The Guided Local Search (GLS) is one such iterative improvement meta-heuristic approach successfully applied to QAP. It has been applied to combinatorial optimization problems including SAT, MAX-SAT, Vehicle Routing, Workforce Scheduling, Radio Link Frequency Assignment, Traveling Salesman, and Function Optimization. It operates as an algorithmic framework utilizing a problem-specific local-search in navigating a modified solution landscape induced by an augmented objective function. This augmented objective function is modified each time the local search reaches a local optimum by penalizing select features of the solution.

This paper proposes the following modifications to GLS:

1. Incorporation of an iteration constrained aspiration criterion designed to further diversify the search similar to the mechanism seen in Taillard’s robust tabu search [4].
2. Evaporation of feature penalties in the augmented objective function to combat too excessive a deformation of the original solution landscape similar to notions borrowed from ant colony optimization [5].
3. Introduction of an intensification policy based on periodic executions of a steepest-descent search on the original objective function [6].

The contribution to the research community is as follows:

* Robust extensions to the guided local search that may be applied to other combinatorial optimization problems
* Innovative and competitive new approach for the QAP.

The paper is structured as follows. In Section II related works for QAP is elucidated. In Section III, the guided local search framework is discussed in more detail. Section IV details the authors’ specific modifications. Section V elaborates on benchmarks, experimentation methodology, results, and analysis. Section VI ends with concluding remarks and future research potential.

# Related Works

Much of the work done on the quadratic assignment problem can be organized into three distinct categories. Due to the intractable nature of the quadratic assignment for more than the most trivial of solutions (), a testament to its inherent complexity, exact algorithms are simply infeasible to apply. Thus many heuristics and meta-heuristics have been successfully proposed and applied to the QAP and have obtained excellent results solving many much larger sized instances to sub-optimality (conjectured optimal values). While heuristics are unable to guarantee performance characteristics such as production of optimal value within distinct time limits, they in practice produce acceptable results in vastly less time.

Exact algorithms historically applied to QAP include branch-and-bound procedures, dynamic programming, and cutting plane techniques, with branch-and-bound being the most pervasive and successful in the literature [7], [8]. Heuristic algorithms include constructive, limited enumeration, and improvement methods, with most of the heuristics falling under the latter. Constructive methods either construct entirely new promising solutions at each iteration in the search process or they may complete partial solutions, most likely by utilizing adaptive memory to enhance construction. Limited enumeration methods are simply branch-and-bound procedures subject to various constraints to produce good solutions in feasible time. Improvement methods, also known as local search algorithms, start with a single initial solution and perform a series of modifications to traverse the solution space. Many schemes may employ elaborate mechanisms to traverse past local optima.

Meta-heuristics are a class of heuristics that may either be constructive or improvement heuristics but differ in that they are general heuristic frameworks that are problem-agnostic and have been found to perform well on a large range of combinatorial optimization problems. Meta-heuristics fall under two main categories – those based on theoretical and experimental considerations or on natural processes such as the foraging behavior of ant colonies [5], [9] (Ant Colony Optimization) or how Darwin’s natural selection evolves a more superior population (Genetic Algorithm) [8]. Successful applications of meta-heuristics pertaining to QAP that fall under the former category include Simulated Annealing (SA) [10–13], evolution strategies [14], Genetic Algorithms (GA) [15–18], scatter search (ScS) [19], Ant Colony Optimization (ACO) [5], [20], [21], and neural networks and markov chains. Those in the latter include Tabu Search (TS) [4], [6], [22–26], Greedy Randomized Adaptive Search Procedure (GRASP) [27], Variable Neighborhood Search (VNS) [17], [21], Guided Local Search [28], [29] and hybrid heuristics [30–32].

Hybrid algorithms in most cases perform better than other meta-heuristic approaches when applied to QAP [8]. These include an ACO periodically populating a GA’s pool of solutions [31], or an “evolutionary reactive tabu search augmented with selection and recombination operators” [32]. Most constructive meta-heuristics apply a local improvement heuristic or meta-heuristic to fine-tune crudely constructed solutions including a naïve steepest descent search, Lin Kernighan variable neighborhood search, or short runs of robust tabu search (RoTS) [5]. What follows is an exposition on the state-of-the-art meta-heuristics for the QAP, or those that have found best-known values in the QAPLib [33].

## Tabu Search

The Tabu Search (TS) algorithm introduced by Glover has been applied to problems in routing, scheduling, graph partitioning, location and allocation, design, production, logic and artificial intelligence, and telecommunications “yielding results that significantly surpasses that obtained by methods previously applied” [34]. When applied to QAP, it has also demonstrated its prowess. Tabu search is a local improvement algorithm that is able to traverse past local optima by means of a tabu list which forbids past moves (swaps or exchanges). At each iteration it finds and takes the single best move in its current local neighborhood that is not forbidden or satisfies some aspiration criteria such as improving the best-known solution seen thus far. Tabu search was first applied to QAP in a method called Tabu Navigation [24], obtaining better results than had been seen previously in various simulated annealing approaches. However, it utilized multiple runs of the basic tabu search with varying parameters between each run.

The Robust Tabu Search (RoTS) [4], [35], which is prevalent in many state-of-the-art hybrid meta-heuristics, prohibits moves where both interchanged facilities occupy locations previously recently occupied rather than simply forbidding index pairs. The approach is not only simpler but more robust, finding sub-optimal solutions for most small and medium-sized problems of up to 64 [4] and producing best-known values for the larger structured problems of Skorin-Kapov and Taillard as found in QAPLib (*Tai80b*, *Tai100b*, *Sko72*, *Sko90*) [33]. It does this by means of a single solution pass, efficient data structures and objective function evaluation, additional aspiration criterion to force overdue swaps, and dynamically varying tabu length to mitigate cycling and escape basins of attractions [4].

The iterated tabu search (ITS) [6] is a modified tabu search that disregards the iteration-constrained aspiration criterion while retrofitting certain intensification elements, including periodic executions of a steepest-descent local search, probabilistically allowing all swaps regardless of its tabu status, and reduction of tabu periods of forbidden elements on reaching new best local optima. It then utilizes multiple short runs of this in a larger framework designed to perform delicate chained mutations of the last local optima reached between runs. The results are phenomenal, establishing new best-known values for *Tai50a*, *Tai80a*, *Tai100a* and generating better results in faster time over most QAPLib instances when compared to RoTS, ReTS [25], ETS [23] , and GEN-4[18].

The same author of ITS proposes the Enhanced Tabu Search (ETS) [23] using the same modified tabu search and iterated search mechanics but this time with various other more disruptive mutation schemes when stagnation has occurred. ETS outperforms both RoTS and GEN [15] while finding the current best-known value for *Tai60a*; however, ITS still supplants ETS due to its innovative approach to mutation.

## Genetic Algorithm

Genetic algorithms operate on the premise of evolution and natural selection whereby certain anomalies in population create elite members who outlive their counterparts. The basic framework is as follows: An initial population is seeded and evaluated. Offspring are created using crossover and mutation mechanics, and culling excises inferior members to foster an elite population.

A hybrid genetic algorithm (GEN) [15] is proposed that seeds the initial population with the output of RoTS ( iterations), and utilizes short RoTS runs () as the mutation mechanism performed after a crossover. Parents are chosen randomly according to rank, and crossover is performed by copying overlaps in the parents and partitioning the rest. Given enough time the algorithm is not only superior to a purely genetic scheme but is shown to improve the basic RoTS on Skorin-Kapov instances, establishing the current best-known solutions for *Wil100* and *Sko100* instances [15], [33].

GEN-3 [17] modifies GEN by swapping RoTS with a simple first-descent algorithm, while utilizing the same crossover method. The first-descent algorithm simply considers n randomly ordered distinct swaps and executes a swap if the solution will improve; this process is repeated twice. In fact it was found that, for the more structured instances of *Tai-b* and *Sko*, GEN-3 surpasses GEN; however, for unstructured instances neither GEN, GEN-3, nor HAS-QAP can compete with tabu searches [17]. GEN-3 currently holds the best-known values for *Tai150b*, the largest of the structured instances in QAPLib.

## Ant Colony Optimization

Ant colony optimization algorithms are nature-based meta-heuristics inspired by the foraging behavior of ant colonies [5], [9]. Ants lay pheromones in their search to food sources, while other ants probabilistically follow paths weighted by pheromone strength. Eventually, shorter paths to the food source is reinforced and therefore traversed the most. The min-max ant system (ANT) [5] is applied to QAP whereby each facility/location assignment possesses a corresponding pheromone restricted between upper and lower bounds. Ants randomly traverse edges in constructing a valid solution guided by the pheromones and the resulting either global or iteration best solution is used to reinforce pheromones along with evaporation of all pheromones taking place. Utilizing a 2-opt best-exchange local search, ANT was found to vastly outperform RoTS and ReTS on structured instances, finding the current best-known value for *Tai256c*.

Our approach extends GLS’ and its two basic extensions by more advanced aspiration criteria over the basic best improvement extension and partial, full, and repeated penalty relaxation schemes over the naïve random moves extension.

# Guided Local Search

The Quadratic Assignment Problem is classically known in the literature as one of the hardest of the hard combinatorial optimization (CO) problems. It has been proven to be intractable for sizes above n>25. Due to the complexities inherent in this tough problem, exact algorithms applications are simply intractable and heuristics must be used. Thus, QAP has been one of the archetypal combinatorial optimization problems for benchmarking, testing, refining, and development novel heuristics and meta-heuristics approaches. This is also facilitated by means of the excellent up-to-date centralized set of benchmarks, results, and references used extensively throughout the literature, known as the QAPLib [33].

## Local Search

Due to QAP’s hardness and the need for heuristics, one of the most popular heuristic scheme used in the literature is the improvement heuristic. These operate as follows: Given a random solution in the solution space of potentially different solutions – one for each permutation in a combinatorial optimization problem – an improvement heuristic observes a limited set of neighboring solutions and chooses either the first improving solution or the best improving solution. The fitness or quality of the solution is simply the evaluation of the **objective function** for QAP as shown in (1) when input with the solution. Since QAP is a minimization problem, a solution improvement occurs when the new solution resolves to a lower objective function value than the current solution. The goal is to quickly find the most minimal solution, known as the **global optimum**, in tractable time. The simplest of **neighborhood structures** is based on 2-Opt and is the set of all solutions obtained by swapping two elements in the current solution. This can be generalized to the case of k-Opt, or the set of solutions obtained by swapping k elements in the current solution. Traversing increasingly complex neighborhood structures increases resolution and accuracy, while suffering from exacerbated time complexity to potentially negate the effectiveness of any meta-heuristic frameworks utilizing the local search. Once a favorable neighboring solution is found, the local search “moves” or performs the needed swap to obtain that solution. In doing so it has traversed the **solution landscape**. This landscape may be thought of as a mountainous region with hills, craters, and valleys and is induced from the objective function, set of all solutions, and a distance measure [3]. This process recurs until no neighboring solution can be found for which an improvement can be made. Once this occurs, the capstone issue of all local searches arises – namely, how to escape local optima. The solution at this point is known as a **local optimum**.

Many attempts at escaping local optima has been given in the literature and these usually occur by wrapping a larger more complex framework around the basic improvement local search. This larger framework is known as a meta-heuristic and can be either constructive or improvement-based. It is constructive if it uses the locally optimal solution, or the output from the local search, updates an adaptive memory, and strategically surveys other areas of the solution space by generating a new solution based on the adaptive memory. This new solution is then run through the local search and brought to its local optimum. On the other hand, a meta-heuristic is improvement-based if, instead of constructing new solutions, it contains mechanisms to traverse past the current local optimum by either performing increasingly intense mutations of the solution, known as perturbations or jolts, allowing for a time traversal to worsening solutions, or temporarily pruning the local neighborhood structure by forbidding certain swaps from being made. The best of the remaining allowed swaps may be a worsening solution; however, the process is temporary and allows escaping local optimum. Examples of improvement meta-heuristics used in the literature are simulated annealing (SA), tabu search (TS), and guided local search (GLS). In the following we survey GLS, including a discussion on its background, mechanics, and implementation.

## Background

Since the Guided Local Search (GLS) is a meta-heuristic, or a problem-agnostic framework adaptable to multiple combinatorial optimization problems, GLS has indeed been applied to many such problems. In addition to QAP, GLS has been applied to the vehicle routing problem, the radio link frequency assignment problem, function optimization, traveling salesman problem, SAT, and weighted MAX-SAT [29]. GLS is a member of the class of meta-heuristics known as dynamic local search, and can be seen as a direct descendent of the tabu search in the hierarchical classification of meta-heuristic approaches. It was devised and generalized from the “GENET neural network for solving constraint satisfaction problems and optimization problems” [29]. It was further generalized from the “min-conflicts heuristic repair method by Minton et al.” developed again for constraint satisfaction problems [36]. Recently GLS has been used with a genetic algorithm as the local search method and successfully applied to the processor configuration problem, the generalized assignment problem, and the radio link frequency assignment problem [29]. GLS is related to an area of search theory focused on distributing search effort [36].

The Guided Local Search (GLS) is a meta-heuristic with the goal of quickly producing excellent, even optimal, results within tractable time. To be precise, it is an improvement-based meta-heuristic operating a local search heuristic over an augmented solution landscape to guide its search to favorable unexplored areas of the solution space. Local optima is bypassed by selective permanent penalization of features in the solution, thus modifying the augmented objective function and implicitly the augmented landscape such that the solution is no longer considered a locally optimal solution when evaluated by the augmented objective function. The ingenuity lies in the method used for selection of the subset of features to penalize in a solution. Penalizing all features in a solution too harshly restricts features that may be inherent in the global optimal solution, whereas penalizing the wrong features may inhibit the exhaustive exploration of the solution space, which is crucial in locating the global optimum. Due to the novel penalization scheme, effective exploration and guidance to favorable solution space areas is made possible – in other words, search effort is frugally expended. The following details the individual mechanics of GLS to make this possible.

## Mechanics

For any combinatorial optimization problem that GLS is applied, the distinct features of a solution of the problem must be cited. For the traveling salesman problem, for instance, a feature might be each of the edges jutting from the city connecting it to the other cities. The total set of features would be the set of all distinct undirected edges in the complete graph. For the QAP, a feature is whether a particular facility is placed at a particular location in a solution. From this definition it is clear that there are features in any QAP solution. GLS stores the integer penalty of each feature in an matrix. All penalties are initialized to 0. Since the augmented objective function is calculated based on these penalties, modification of any entry of the matrix results in an augmentation of the solution space. The augmented function is simply the value of the objective function in addition to the sum of the penalties of all features exhibited by the solution multiplied by a scaling factor. The augmented objective function is defined formally below:

(2)

*g* is the original objective function defined in (1), is the penalty of feature , is an indicator function which outputs 1 if the feature is present in the solution, and 0 otherwise, and is a parameter of GLS, which varies the intensity of the augmentation of the landscape. A higher value of exhibits more disruptive augmentation resulting in coarser searching and higher diversification, while a lower value results in greater intensification at the cost of reduced exploration. In practice, this selection of affects the efficiency of the search and GLS performance is relatively insensitive to the choice of for many problems [36]. Lastly, the coefficient is a value sensitive to the given problem instance and is derived from the average change in the objective function value for a single move or swap in a solution:

(3)

Recall that and are the distance and flow matrices, respectively.

The specific functioning of GLS is as follows. The local search (LS) utilized is a simple steepest descent algorithm. The local search is fed the current solution and the augmented objective function given in (2). The neighborhood structure used is 2-Opt. The local search evaluates each of the neighboring solutions using the augmented objective function and traverses or moves to this solution only if it is better than the current solution again based on the augmented objective function. This process is repeated until no improving solution is found. At this point, the single feature with the highest utility in the locally optimal solution is penalized by increasing the corresponding entry in the penalty matrix by exactly 1. The utility of a feature is calculated as follows:

(4)

The cost of featureis one of the pivotal mechanics of GLS to not only traverse past local optima but guide the search to favorable areas of the search space. The cost of feature is defined as the proportion of the objective function for which the feature contributes. By penalizing features with the highest costs, solutions in the future possessing such features are more likely avoided. In this way, the future search is guided to more favorable locations for which the majority of the features are of lower cost, and thus the resulting objective function will also be of lower cost. In a minimization problem such as QAP, doing so is highly desirable. Of equal importance, stemming from the denominator in equation (3), is the fact that features penalized in the past are less likely to be penalized in the future. This provides a more equally distributed exploration of the solution space. The cost of feature of solution in the case of QAP is calculated the following way:

(5)

Note that the sum of the costs over all features in a solution is precisely the objective function (1). Once a single feature with the highest utility is penalized, the augmented objective function is implicitly modified. The process detailed above is repeated on the current solution with the newly modified augmented objective function, which is no longer a local optimum due to the penalization. This process is repeated until some stopping criterion is met, such as number of iterations, runtime, or the relative change in successive solutions.

The last implementation detail is the fact that the change in augmented objective function cost from the current solution to a neighboring solution can be quickly calculated iteratively based on the last change in amortized , making searching the entire 2-Opt neighborhood possible in amortized .

, (6)

where is the change in augmented cost of permutation after the elements and have been swapped, is the penalty when the th element of permutation is assigned to value , and is the change in original objective function cost, which can be found in [28] with efficient implementation in [6]. Below is the pseudo-code for GLS:

{ **while** (not termination criteria){  
 **foreach** ( in {1..n}), where is maximized  
   
 }  
 **return** }

1. Pseudocode for GLSQAP [28]

{**do** {  
 in s.t. is minimized  
 // is the 2-Opt neighborhood  
 // see (6)  
 **if**   
 **if**    
 **else**    
 **if**   
 } **while**   
 **return** }

1. Pseudocode for LS in GLSQAP [29]

## Extensions

Two important extensions to the GLS were developed in Mills’ doctoral dissertation in [29]. Both were shown to improve the insensitivity of while producing on average more robust results. It was concluded that the inclusion of both enhancements should be confidently incorporated into the basic GLS when applying to other combinatorial optimization problems. The first extension is known as the best improvement iteration criterion borrowed from concepts in Tabu Search, while the second extension employs randomly moving to a neighboring solution, disregarding its quality, a low percentage of the time.

### Best Improvement Aspiration (AspBest)

GLS suffers from the problem of permanent penalization or augmentation of the landscape; thus, in a problem where one is trying to find a minimal cost solution from the original objective function, sole operation on an augmented landscape may cause undesirable side-effects. One approach used to combat this is the so-called Best Improvement Aspiration Criterion. This borrows heavily from the concept found in Tabu Search, where any forbidden swaps are temporarily ignored if swapping the two positions results in a new best solution of all those found so far, known as the **global** **best** **solution**. Extrapolating this to GLS, naturally this would mean ignoring penalties and therefore the augmented landscape any time a certain swap produces a global best solution.

By using the best improvement aspiration criterion, GLS was found to improve solution quality especially when large values of are used [28]. This is due to the fact that larger values mean penalties imposed have a greater effect in the search process. The more penalties that are imposed the more likely the global optimal solution is ignored when solely traversing an increasingly deformed augmented landscape. Secondly, the extension makes GLS more insensitive to larger values of . Moreover, more better-than-previous solutions were found per run. Further investigation found that these improvements did not hinge on the fact that the penalty was arbitrarily ignored a percent of the time but only when ignored at pivotal points, such as when a global best is discovered. The authors state that “it is precisely when and what aspiration does that is critical in its success” [28].

### Random Moves [28] (RandMove)

The second modification employs moving to a random solution in the neighborhood of the current solution a certain percent of the time. When this occurs, the augmented function is effectively ignored. It addresses another fundamental problem in GLS, namely, that of an impedance in diversification or the inability to search certain sections of the search space due to penalties imposed. Equipping GLS with this mechanic intensifies diversification especially for small values of . This was claimed based on an increased average entropy of facility-location assignments during the search with random moves for all values of . Moreover, the extension gave a reduction in the amount of repeated solutions when is set too low. Based on these results, GLS with random moves produces higher quality solutions stemming from a more diverse search and an enhanced ability to escape local optima.

# Extensions

This paper focusses on a number of extensions aimed to amend or ameliorate certain issues inherent with Guided Local Search. The motivations for researching and implementing extensions arise in response to the fundamental impediments or weaknesses of GLS.

The overarching motivation stems from the major weakness of permanent penalization. As was mentioned, Guided Local Search operates on an augmented solution landscape with penalties being imposed after each run of the local search, or after each local optima is found in the augmented landscape. This penalization process is permanent and therefore the augmented landscape continually undergoes a deformation. Because the entire search operates on solely this landscape, it seems obvious that as time goes on the search becomes increasingly obfuscated with respect to the original objective function for which a minimum is solely sought. For instance, assume that the global optimal solution possesses a certain high cost feature, which GLS will be more likely to penalize. This penalization, especially since it is permanent, means that this solution will be missed during the subsequent search due to the augmented landscape misrepresenting the original landscape. Even with the utility favoring penalization of features that have not been penalized previously, there is still a large chance that this global optimum is missed.

Furthermore, even if the global optimum should possess only low cost features, the path en-route to this global optimum may be riddled with solutions possessing high cost features. Because these are all more likely to be penalized, the augmented landscape may prevent the traversal of the entrance to the area which contains the global optimum. Penalization is still the quintessential part of GLS, for without it, there would be no means for bypassing the local optimum found at the end of the local search execution. However, one of the major motivations for the extensions discussed below hinge on the fact that perhaps this penalization need not be permanent and that this landscape can in some way be reformed or molded back into the original landscape over time through some reduction or evaporation in the penalty values. This motivation was inspired from ant colony optimization where pheromones are evaporated over time in order to *forget* poorly placed pheromones over poor solutions.

The second weakness inherent in GLS is lack of a true diversification mechanism. One way GLS may address this is to set to an excessively high value. This way when a feature is penalized it vastly restricts the area around that solution in the landscape and more total landmass of the solution landscape is explored. Doing so, however, exacerbates the problems discussed above from permanent penalization. Furthermore, doing so will cause GLS to be unable to properly intensify or exploit the area around good solutions to find any good solutions nearby because the search becomes too coarsely grained. Thus it seems that the only mechanism in place to diversify or intensify stems from . However, it seems that should be used solely for minute tweaking of the augmented landscape and set as low as possible to accomplish only bypassing the current local optima and deform or restrict as little of the neighboring solutions as possible. This allows for intense intensification around the area if good solutions exist nearby to the current local optimum. Thus small values of are better than large ones to foster exploitation. However, doing this means exploration of the search space is greatly impeded. Thus equipping GLS with additional diversification mechanisms is highly desirable.

The various extensions investigated include implementation of additional aspiration criteria over the basic best improvement criterion, probabilistically ignoring the augmented landscape and reverting back to the original landscape for a time, tests to measure the effectiveness of the GLS’ utility scheme for selecting features to penalize including the amount of penalization performed at each step, varying dynamically based on a non-uniform random distribution, introducing noise into the evaluation of the augmented objective function, and other various methods aimed at addressing GLS’ flaws.

## Extensions Addressing Diversification

The following extensions address GLS’ inability to properly diversify during the search process when using necessarily small values of to ensure proper intensification. The schemes investigate an additional aspiration scheme to overdue facility/location assignments and injection of noise into each evaluation of in order to promote diversification.

### Iteration Constrained Aspiration (AspLate)

An iteration constrained aspiration criterion borrowed from Robust Tabu Search was investigated. In Robust Tabu Search (RoTS), it was found that performance hinged most pivotally on this aspiration criterion rather than simply temporarily forbidding swaps [29]. The operation of this aspiration criterion is as follows: Any time a swap between two elements (facilities) in the current solution is performed, the current iteration number is recorded for both facilities at their new location in an matrix . More precisely, let ), where ’ is the new solution after swapping elements at index and in solution . , where is the number of swaps since the beginning of GLS. During the normal functioning of LS (see Fig. 2), is ignored and a swap is enforced if either of the facilities involved in the candidate swap has not been placed at the locations in the swap for a certain extended number of past iterations as examined based on . More precisely, a swap on is invoked if , where is the current iteration (number of swaps since the beginning of the search). Since there may be multiple such swaps, the swap producing the minimal original objective function cost from (1) is used. This mechanism plays a role in guiding the solution to novel areas of the solution space previously unexplored and further preventing any search stagnation or cycling around a particular location. This is referred to as a *hard swap* that forces the search to move certain places, in contrast to an optional *soft swap* scheme that could be used to merely strongly suggest via incentives such actions [29]. See *Late* and *AllLate* variants below an implementation of the latter scheme.

### Augmented Objective Function Noise (Noise)

The next extension addressing the problem of lacking diversification is referred to as *Noise*. The notion of this extension is simple: Inject slight uniformly randomly generated noise into the evaluation of the augmented objective function in order to investigate the possibility of moving to novel areas of the search space which might have been prevented due to penalization of intermediate locations in the landscape. The precise modification of the resolution of given in (6) is as follows, where is uniformly randomly generated, and varies the intensity of noise:

(7)

### Multiple Threads of Execution in Lockstep (Multi)

This extension investigates multiple concurrent runs of GLS. To be precise, initial solutions are randomly generated. *LS* is run on each of the solutions to obtain unique local optima. All optima update the same matrix of penalties and thus share . The process repeats on . Motivation of this is to ascertain if multiple concurrent solutions dispersed throughout the solution space not only aid in diversification but offset penalization establishing undesirable barriers en-route to novel areas.

## Extensions Addressing Permanent Penalization

The primary weakness of GLS is its operation on an augmented landscape with no means to forget penalization of poorly chosen features, even if they were of highest utility. The extensions range from an aspiration scheme to force moves to high quality solutions, regular intervallic relaxation of the penalties for one or a sequence of moves, and evaporation of penalties at various granularities.

### Best Improvement Aspiration (AspGood)

This aspiration criterion is an extrapolation from the basic best improvement aspiration criterion. Due to the success of ignoring penalties if a certain swap results in a new global best solution inherent in the original aspiration criterion, one would expect additional performance gains from aspiring using the best solutions seen so far. In other words, if a swap produces a solution using the original objective function that is better in quality than the best seen so far, the swap is forced. Notice through all of this that the penalties and the corresponding augmented function is ignored for all these solution cost evaluations and subsequent forced swaps. One caveat is that forced moves to the actual best solutions are forbidden to prevent cycling. This extension should theoretically increase exploration of higher quality areas of the search landscape that would be been ignored due to permanent penalization and sole operation on an augmented landscape.

### Best Objective Function Move (BestMove, BestMovePr)

This approach deals with regular relaxation of the penalties and reversion back to for solution evaluation for a single move. In other words, at a set interval or probabilistically, is ignored, and instead a move to the neighbor minimizing is forced. The motivation of both variants are in order to investigate if relaxation of penalties for a single move might guide the search to novel areas and discovery of higher quality solutions.

### Steepest Descent (SDAlways, SDInterval)

This extension is similar to *BestMove* and *BestMovePr*; however, it employs repeated contiguous applications of both *BestMove* on thus ignoring for a time. This process is equivalent to running a standard steepest descent local search on . *SDAlways* always occurs after each run of *LS* from Fig. 2 and directly prior to penalization. *SDInterval* executes steepest descent periodically every static number of swaps/iterations. Motivation of this is to investigate if a more powerful relaxation does a better job at intensification.

### Evaporation

Instead of occasionally relaxing the penalties, evaporation permanently reduces some or all of the penalties every so often. Doing so should forget badly penalized solution features, especially those existing in the global optimum. Furthermore, it should aid in discovering novel areas of high quality solutions inaccessible behind barriers of excessively penalized intermediate solutions. In other words, barriers are gradually decayed overtime. The following evaporation variants operate on different levels of granularity, either every swap or every number of local optima, and either reduce all or a subset of penalties.

The first set of variants pertains to partial penalty reduction. Their granularity is per local optima. Their operation is as follows: Every time a feature is penalized in the penalty matrix , one or more features of the current local optimum has a reduction of penalty. Let feature in local optimum be the worst of ’s features, or the one possessing the greatest penalty value. *Worst* reduces this penalty by some amount less than the penalty amount (1 by default). partitions the reduction amount non-uniformly among all features weighted by their current penalty values. is an alternate scheme to address GLS’ diversification weakness. The feature that is latest, or possesses the minimal entry in the swap matrix (see *AspLate*), is reduced in penalty. This investigates adding incentives to GLS to unobtrusively favor overdue swaps at some future point [29]. reduces all features’ penalties weighted by its lateness relative to the current iteration.

The second set of variants reduces all penalties at varying levels of granularity. All variants multiply all entries in by , where is a static penalty reduction amount. The differences however lie in when and where this reduction is performed. *EvapOnSwap* performs a finer reduction of penalties every swap. *EvapOnInterval* performs a larger reduction periodically every number of unique local optima, but only after penalization of the optima (see Fig. 1). *EvapOnImprovement* reduces penalties whenever a new global best solution is discovered. *EvapSinceImprovement* reduces penalties only after a set number of iterations of no improvement in the global best solution. *EvapSinceImpDecay* is a derivation of prior variant with the additional mechanic of decaying or dividing by a static amount each execution of the variant. is initialized to 1.0 and decays thereafter on successive applications.

## Other Extensions

The following details other extensions testing certain aspects of GLS to see if they may improve its functioning. These include dynamic variation, different penalization schemes and amounts, and lastly a new search designed to use GLS only for exploration.

### Dynamic (LambdaPow)

This extension entails calculating dynamically every time it is used to calculate . It was designed to explore the possibility that varying might make the search more robust in terms of exploitation and diversification since low (high) values promote the former (latter). This approach was inspired from RoTS’ non-uniform, random generation of its iteration-constrained aspiration length parameter [37]. is varied non-uniformly favoring values closer to . is random. is the measure of non-uniformity.

(8)

### Penalization Schemes

This set of extension explores different penalizations schemes and amounts. *PenaltyUtil* uses differently sized penalization amounts beyond the default 1.0. *AllUtil* extends this by distributing the penalization amount to all features in the local optimum weighted by . *Cost* ignores utility and simply penalizes the maximum cost (5) feature of . *AllCost* extends this by distributing the penalization amount weighted by cost to all features.

### GLS as Explorer

This extension is motivated by the fact that since GLS operates on an increasingly inaccurate augmented landscape, one should use GLS only as a secondary method for exploration, while primarily using another search that operates solely on the original landscape for a more accuracy. *Steep* uses a steepest descent on to take a solution to its local optimum, penalty matrix is updated, GLS is run for swaps, , and the process is repeated. is reset when a new global best is found. *SteepDist* operates on the notion that since should technically assert that the solution output from GLS is exactly this hamming distance away from the input solution. *SteepDist* modifies *Steep* by running GLS until . *SteepTS* and *SteepDistTS* use short runs of RoTS instead of steepest descent.

# Results

This section is organized as follows: The test configuration for each variation along with the set of benchmarks used is presented. Lastly, results are presented, grouped by the weakness in GLS it addresses. When a result is presented, brief analysis is given on reasoning and significance.

## Configuration

Algorithms were developed in C++ and compiled using GCC 4.4.3 with the –O3 flag. They were run on a collection of Sun Ultra 24 workstations running SunOS 5.10 and Intel Core 2 Duo E8600, 3.33GHz processors with either 4 or 6 GB of ram. Code for RoTS was adapted from [37] and for GLS from the literature [28], [29]. GLS and modifications were tested against *GLSSolver*, a program provided by these authors, and give equivalent results to assert validity. All code and results can be found on **qap.googlecode.com**. Benchmarks were taken from *QAPLib*  [33], a QAP library pervasively used throughout QAP literature. The 9 instances from QAPLib used were the most complex and well survey the 4 unique instance types of *QAPLib*: Type I are unstructured, uniformly randomly generated distance and flow matrices, and hardest in practice to solve. Included are *tai80a* and *tai100a*. Type II are non-uniform (structured), random flows on grids and Manhattan distances between squares. Included are *sko90, sko100a, wil100, tho150*. Type III are strongly structured, real-life instances with sparse flow matrices. These are easiest to solve in practice. Included are *tai256c*. Type IV are structured, larger real-life-like instances with distances between points scattered non-uniformly over a circle, and non-uniform generation of flows. Included are *tai100b* and *tai150b*. Each instance in *QAPLib* possesses a best known value (*BK*) as produced by a state-of-the-art meta-heuristic. Relative error is measured as such:

(9)

Algorithms were averaged over a number of runs on each of the 9 instances cited above. Results mentioned below present the *%RelativeError* implicitly summing these 9 averages. GLS variants use and utilize *AspBest,* unless otherwise stated. Runs are exactly 15 minutes long. Base algorithms taken from the literature were run 15 times per instance. The GLS variants presented in this paper were run 5 times per instance. The detailed breakdown of results for each instance including metrics such as averages, standard deviations, minimums, minimum count, average run-time of last found global best, can be found on **qap.googlecode.com**. What follows are the results of the base algorithms, vital for benchmarking against all results presented later.

## Base Algorithms

The base algorithms include RoTS [37], GLS [28], GLS+*AspBest*+*RandMove* (GLS2) with optimal parameter settings suggested in [28], GLS+*AspBest* (GLS3), randomly constructed solutions enhanced with steepest descent (WALK1), fast descent found in [17] (WALK2), and Lin Kernighan (WALK3). RoTS gave excellent results of 2.22. GLS and base variants produced modest results. GLS gave 3.992, GLS2 gave 3.639, GLS3 gave 3.869. The random walks gave poor results, which is to be expected due to their naivety: WALK1 gave 9.359, WALK2 gave 22.726, and WALK3 gave 8.41. Note that the best of GLS variants before extensions presented in this paper is precisely GLS2 at **3.639**. Any value above GLS3’s 3.869 is considered a satisfactory improvement since all variants directly extend GLS3.

## Extensions addressing Diversification

### Iteration Constrained Aspiration (AspLate)

*AspLate* was run with a variety of different aspiration lengths to tests its efficacy. Too low a value of achieved very poor results of 6.152-14.594. Results steadily grew better until reaching a minimum at with **3.499**. Past this point solutions steadily grew to 3.98. Finally, *AspLate* was tested with a variety of while keeping . The optimal was producing 3.549. It seems that *AspLate* is a viable means to achieving proper diversification in GLS by forcing overdue swaps to occur during the local search as was initially conjectured.

### Augmented Objective Function Noise (Noise)

*Noise* was tested with a variety of noise intensities Unfortunately, *Noise* produced unsatisfactory results. When no noise was applied (), the results were equivalent to GLS3 at 3.86. When noise was applied, quality was 13.399-17.442. This may have occurred because GLS requires operation on a static landscape unchanged during a local search.

### Multiple Threads of Execution in Lockstep (Multi)

*Multi* was implemented to explore if simultaneous searches across different parts of the solution landscape might enhance diversification and offset penalization barriers. was varied between [1,10]. At , the algorithm devolves into GLS3. At greater values of , values were generally uncorrelated in the range 3.916-4.312. The reason for this might have been negative consequences stemming from sharing the same penalty matrix. It is conjectured that not sharing them will produce more robust values at least better than GLS3 since its minimum was 2.696, or 1.173 lower than 3.869.

## Extensions Addressing Permanent Penalization

### Best Improvement Aspiration (AspGood)

*AspGood* was run with a variety of . Results across the entire range were sporadic and uncorrelated between 3.657-4.161. The best result of **3.657** is produced from setting . It appears that using marginally achieves better results than the simpler *AspBest*, where . Hence, it appears that allowing multiple forced moves to elite solutions enhances exploitation due to longer operation on the original landscape.

### Best Objective Function Move (BestMove, BestMovePr)

*BestMove* was tested on a variety of intervals, ]. Results were slightly correlated favoring smaller around with values of 3.673-3.89. Elsewhere, results varied from 3.762-4.051. The best value was producing 3.673. *BestMovePr* was tested with probabilities . Results were variable (3.614-4.013) until when results degenerated (4.727-4.865). The optimal values found were giving **3.614**. It can be concluded that regular relaxation of the penalties for a single move enhances GLS3. Furthermore, using intervals obtained more stable results than probabilities.

### Steepest Descent (SDAlways, SDInterval)

*SDAlways* has no additional parameters. Results were an unsatisfactory 6.332. However, *SDInterval* produced much better results. *SDInterval* was run on 25 different intervals . Results grossly improved from giving 13.716-5.525. Beyond , values steadied with weak positive correlation in 3.565-4.191. The best result of **3.565** was found when . Clearly running SD too often is unwise; however, on low intervals SD is found to improve GLS. It appears that regular relaxation of the penalties for a sequence of moves (*SDInterval*)is more advantageous than a single move (*BestMove)*.

### Evaporation

The results for partial penalization variants follow. All variants used the default penalization amount of 1.0, with an evaporation amount of .9. *Worst* gave 4.016, *AllWorst* gave **3.455**, *Late* gave 4.291, and *AllLate* gave 3.675. In both cases, the *All* versions, where all penalties were reduced weighted by some criteria, either penalty or lateness, produced much better values than their counterpart. It appears that addressing penalization rather than diversification when evaporating proves more advantageous.

The results for full penalization variants follow. The finest grained evaporation scheme, *EvapOnSwap*, was run with varying producing poor values of 6.075-17.301. The more intense the reduction the stronger a result’s quality degraded, at an exponential rate. *EvapOnInterval* was the most aggressively tested variant with 70 unique tests for varying intervals of varying reduction strengths . For all , using too low an interval produced excessively poor results of 13.111-8.343. Results sharply increased in quality and levelled off or slowly decreased in quality for increasing over all . This taper curve was more pronounced the smaller became. In other words, larger needed larger but generally produced better, more stable results. The range of values were 3.23-13.11. Best results were 3.23-3.472 for 8 tests over . More precisely, yielded **3.23**, and yielded **3.241**. *EvapOnImp* was tested in . Results steadily degraded from 4.417-6.531 until a sharp improvement occurred at of **3.739**. 30 tests were run on *EvapSinceImp* for with 7 tests producing 3.35-3.604. Results were better than *EvapOnImp* with **3.35** found at . 40 tests were run on *EvapSinceImpDecay* over , where is the decay rate. Generally all results were worse than *EvapSinceImp* and no better than 3.557 when .

Based on these results, a roughly grained evaporation schedule is better than a finer one, with quality more pronounced for larger intervals at larger penalty reduction rates, up until a point. Moreover, one can assert that GLS suffers from a permanent penalization weakness whereby evaporation to gradually reform the original landscape and forget poorly penalized features is vital to a more robust GLS.

## Other Extensions

### Dynamic (LambdaPow)

A variety of tests were run for producing poor results (5.438-10.038). It is conjectured that either GLS requires static or that a different skewing curve should be chosen producing no smaller but still favoring , similar to RoTS’ aspiration curve.

### Penalization Schemes

Let be the penalization amount, be 1.0 unless specified. *AllUtil* gave 7.846, *Cost* gave 7.465, *AllCost* gave 8.843, and *Util* for .1, 3] gave 3.922-4.333 showing a parabolic trend minimized at 3.922 (3.934) for . Results were so close to GLS3 (where ), that one can assert that the default utility scheme and amount present in GLS is best.

### GLS as Explorer

*Steep* was run on 34 tests over producing 3.308-3.626 on 6 results, with optimal being **3.308** on . The best of *SteepDist* gave **3.604**, *SteepTS* gave **3.625**, *SteepDistTS* gave 5.278.

# Conclusion

The Quadratic Assignment Problem (QAP) is a problem pertaining to the optimal assignment of facilities to locations to minimize total the total flow/distance according to (1). The best performing heuristics are meta-heuristics of which TS, SA, ACO, and GA are state-of-the-art. Guided Local Search (GLS) is one such meta-heuristic that bypasses local optima by operating on an augmented landscape derived from penalties for each feature (facility/location assignment), which is updated by a novel approach to feature selection for penalizing. This paper details 26 extensions. The best of GLS variants before this paper produced 3.639% relative deviation above best known values over 9 instances from QAPLib. 11/26 variants produced better values from 3.23-3.657. These extensions address GLS’ 2 major weakness, including lack of diversification and permanent penalization. It was shown that addressing both produced substantial improvements over GLS variants before this paper, with addressing its permanent penalization weakness by a coarse-grained full evaporation of penalties at larger intervals of producing the most drastic improvement of 3.23. Future research may include testing combination of variants using particle swarm optimization [6] to optimize parameters, and more elaborate ways of fusing Robust Tabu Search with Guided Local Search.

##### References

[1] P. M. Pardalos, F. Rendl, and H. Wolkowicz, “The quadratic assignment problem: A survey and recent developments,” in Proceedings of the DIMACS Workshop on Quadratic Assignment Problems, 1994, vol. 16.

[2] E. M. Loiola, N. M. M. de Abreu, P. O. Boaventura-Netto, P. Hahn, and T. Querido, “A survey for the quadratic assignment problem,” European Journal of Operational Research, vol. 176, no. 2, pp. 657–690, Jan. 2007.

[3] T. Stützle and M. Dorigo, “Local search and metaheuristics for the quadratic assignment problem,” 2001.

[4] E. Taillard, “Robust tabu search for the quadratic assignment problem,” Parallel computing, vol. 17, pp. 443–455, 1991.

[5] T. Stützle, “MAX-MIN ant system for quadratic assignment problems,” 1997.

[6] A. Misevicius, A. Lenkevicius, and D. Rubliauskas, “An implementation of the iterated tabu search algorithm for the quadratic assignment problem,” OR Spectrum, vol. 34, no. 3, pp. 665–690, 2012.

[7] C. Commander, “A survey of the quadratic assignment problem, with applications,” University of Florida, 2005.

[8] M. Bayat and M. Sedghi, “Quadratic Assignment Problem,” in Facility Location - Concepts, Models, Algorithms and Case Studies, R. Zanjirani Farahani and M. Hekmatfar, Eds. Heidelberg: Physica-Verlag HD, 2009, pp. 111–143.

[9] M. Dorigo, M. Birattari, and T. Stutzle, “Ant colony optimization,” IEEE Computational Intelligence Magazine, 2006.

[10] S. Amin, “Simulated jumping,” Annals of Operations Research, vol. 86, pp. 23–38, 1999.

[11] R. Burkard and F. Rendl, “A thermodynamically motivated simulation procedure for combinatorial optimization problems,” European Journal of Operational Research, vol. 17, no. June 1983, pp. 169–174, 1984.

[12] U. W. Thonemann, “Finding improved simulated annealing schedules with genetic programming,” IEEE Congress on Evolutionary Computation (CEC), vol. 1, no. 1, pp. 391–395, 1994.

[13] A. Misevicius, “A modified simulated annealing algorithm for the quadratic assignment problem,” Informatica, vol. 14, no. 4, pp. 497–514, 2003.

[14] V. Nissen, “Solving the quadratic assignment problem with clues from nature,” IEEE Transactions on Neural Networks, vol. 5, no. 1, pp. 66–72, 1994.

[15] C. Fleurent and J. Ferland, “Genetic hybrids for the quadratic assignment problem,” in DIMACS Series in Mathematics and Theoretical Computer Science, American Mathematical Society, 1994, pp. 173–187.

[16] T. Ostrowski and V. T. Ruoppila, “Genetic annealing search for index assignment in vector quantization,” Pattern Recognition Letters, vol. 18, no. 4, pp. 311–318, Apr. 1997.

[17] E. Taillard and L. Gambardella, “Adaptive memories for the quadratic assignment problem,” 1997.

[18] A. Misevicius, “An improved hybrid genetic algorithm: new results for the quadratic assignment problem,” Knowledge-Based Systems, vol. 17, no. 2–4, pp. 65–73, May 2004.

[19] T. Mautor, P. Michelon, and A. Tavares, “A scatter search based approach for the quadratic assignment problem,” IEEE Transactions on Evolutionary Computation, pp. 165–169, 1997.

[20] T. Stützle and M. Dorigo, “ACO algorithms for the quadratic assignment problem,” New ideas in optimization, pp. 33–50, 1999.

[21] L. Gambardella, E. Taillard, and M. Dorigo, “Ant colonies for the QAP,” 1997.

[22] H. Iriyama, “Investigation of searching methods using meta-strategies for quadratic assignment problem and its improvements,” 1997.

[23] A. Misevicius, “A tabu search algorithm for the quadratic assignment problem,” Computational Optimization and Applications, pp. 95–111, 2005.

[24] J. Skorin-Kapov, “Tabu search applied to the quadratic assignment problem,” ORSA Journal on computing, 1990.

[25] R. Battiti and G. Tecchiolli, “The reactive tabu search,” ORSA journal on computing, no. October 1992, pp. 1–27, 1994.

[26] E. Talbi, Z. Hafidi, and J. Geib, “Parallel adaptive tabu search for large optimization problems,” pp. 1–12, 1997.

[27] Y. Li, P. Pardalos, and M. Resende, “A greedy randomized adaptive search procedure for the quadratic assignment problem,” Quadratic assignment and related …, vol. 40, 1994.

[28] P. Mills, E. Tsang, and J. Ford, “Applying an extended guided local search to the quadratic assignment problem,” Annals of Operations Research, pp. 121–135, 2003.

[29] P. Mills, “Extensions to guided local search,” 2002.

[30] Y.-L. Xu, M.-H. Lim, Y.-S. Ong, and J. Tang, “A GA-ACO-local search hybrid algorithm for solving quadratic assignment problem,” Proceedings of the 8th annual conference on Genetic and evolutionary computation - GECCO  ’06, p. 599, 2006.

[31] L.-Y. Tseng and S.-C. Liang, “A Hybrid Metaheuristic for the Quadratic Assignment Problem,” Computational Optimization and Applications, vol. 34, no. 1, pp. 85–113, Oct. 2005.

[32] J. M. III and W. Cedeño, “The enhanced evolutionary tabu search and its application to the quadratic assignment problem,” Genetic and Evolutionary Computation (GECCO), pp. 975–982, 2005.

[33] F. R. R.E. Burkard, E. Çela, S.E. Karisch, “QAPLib - A Quadratic Assignment Problem Library,” Journal of Global Optimization, 2011. [Online]. Available: http://www.seas.upenn.edu/qaplib/.

[34] F. Glover and M. Laguna, Tabu Search. Boston, MA: Springer US, 1997.

[35] E. Taillard, “Comparison of iterative searches for the quadratic assignment problem,” Location science, vol. 1994, 1995.

[36] M. Gendreau and J.-Y. Potvin, Handbook of Metaheuristics, 2nd ed. Springer, 2010, p. 648.

[37] E. Taillard, “Robust Tabu Search (C).” 2006.