UNIVERSITY OF CALIFORNIA, SAN DIEGO

Evolutionary Algorithms with Local Search for Combinatorial Optimization

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in Computer Science

by

Mark William Shannon Land

1998

The dissertation of Mark William Shannon Land is approved, and it is acceptable in quality and form for publication on microfilm:

Chair

University of California, San Diego

1998

To Jing

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Thanks to my advisor Rik Belew for guiding me through the past few years. Thanks also to Bill Hart (a virtual committee member) and Russell Impagliazzo for their detailed and insightful comments as the dissertation developed. I am also indebted to Sid Sidorowich and ThermaWave Inc. for gently exposing me to the industrial world and funding me for much of my stay. I thank Charles Elkan for some early funding while I was still exploring topics.

I am especially grateful to Chris Rosin for emotional support and constructive criticism of drafts of much of my writing. I thank Thomas Kammeyer, Filippo Menczer, Craig Mautner, and the rest of the Cognitive Computer Science Research Group for helpful comments and interesting conversations along the way. Special thanks to Dan Clouse for his help and the use of his computer when cycles were scarce. Thanks also to Ted Carson for doing extra work to supply me with the Go-With-the-Winners data in Section IV.E.2. Most important, thanks to Jing for walking with me through good times and bad.

VITA

December 7, 1970	Born, Scottsdale, Arizona
1992	B.S., California Institute of Technology
1994	M.S., University of California, San Diego
1998	Doctor of Philosophy University of California, San Diego

PUBLICATIONS

Land, M., SIDorowich, J. J., and Belew, R. K. (1997). "Using Genetic Algorithms with Local Search for Thin Film Metrology." Proceeding of the Seventh International Conference on Genetic Algorithms.

Pollack, J., Blair, A., Land, M. (1997). "Coevolution of a Backgammon Player." Artificial Life V.

Land, M., Belew, R. K. (1995). "No two-state CA for density classification exists." Physical Review Letters 74:25, pp. 5148-5150.

Land, M., Belew, R. K. (1995). "Towards a Self-Replicating Language for Computation." Proceedings of the Fourth Annual Conference on Evolutionary Programming.

FIELDS OF STUDY

Major Field: Computer Science
Studies in Genetic Algorithms
Professor Richard K. Belew

ABSTRACT OF THE DISSERTATION

Evolutionary Algorithms with Local Search for Combinatorial Optimization

by

Mark William Shannon Land
Doctor of Philosophy in Computer Science
University of California, San Diego, 1998
Professor Richard K. Belew, Chair

The goal of global optimization is to minimize (or maximize) an objective function over its entire domain. Heuristic methods such as evolutionary algorithms and simulated annealing are often employed. Alternatively, it is sometimes acceptable to find a *local optimum*, which is as good as all solutions in its neighborhood. Local search methods are comparatively well-understood, and local optima can often be found efficiently even for problems in which global optimization is difficult [66]. Global/local hybrid algorithms combine aspects of both global and local optimization to search more effectively than either global or local optimization by themselves.

Evolutionary algorithms using local search have frequently been applied to problems in continuous optimization with great success. Not only does the addition of local search substantially improve the performance of the evolutionary algorithm, but the hybrid often outperforms other global optimization techniques such as simulated annealing.

In this dissertation we explore the effectiveness of evolutionary algorithms with local search in the *combinatorial* domain. We show that the EA+LS hybrid is more effective for graph bisection than either the EA or LS alone, and that it is competitive with simulated annealing. A new variant of the EA+LS is presented which interleaves the global and local search operators. We find that while crossover is valuable to an EA without LS in this problem, the addition of LS obviates the need for crossover. We also show how appropriate mutation sizes in an EA+LS depend on the size of local basins. These insights point to the possibility of instance-specific heuristics, in which the search algorithm is tailored to specific features of the instances under consideration. We show that Darwinian evolution is as effective as Lamarckian, and is more robust under changes to the genetic operators. In addition, we explore the effectiveness of steady-state vs. generational EAs, different local search lengths, and various local/global ratios.

Chapter I

Introduction

Global optimization is concerned with finding the best possible solution to a given problem. As there are no efficient algorithms to achieve this goal in general, heuristic global optimization methods are often employed. Examples include simulated annealing and evolutionary algorithms (EA).

Alternatively, it is sometimes acceptable to find a *local optimum*, which is as good as all solutions in its neighborhood. Local search methods are comparatively well-understood, and local optima can often be found efficiently even for problems in which global optimization is difficult [66]. Global/local hybrid algorithms combine aspects of both global and local optimization to search more effectively than either global or local optimization by themselves.

Evolutionary algorithms using local search have frequently been applied to problems in continuous optimization with great success [38, 71, 56, 43]. Not only does the addition of local search substantially improve the performance of the evolutionary algorithm, but the hybrid often outperforms other global optimization techniques such as simulated annealing.

In this dissertation we explore the effectiveness of evolutionary algorithms with local search in the *combinatorial* domain. Building on work by Hart [38] regarding continuous optimization, we examine the role of local search and its interaction with the standard evolutionary operators and features of the problem instances. A new variant of the basic algorithm is developed which allows a more complete integration of the local and global search aspects, in addition to greater flexibility in examining the effect of relevant algorithmic parameters. This algorithm is used in experiments on various classes of graph bisection instances. Results are compared to those for other global search techniques, and analyzed in terms of instance features and algorithmic parameters.

I.A Global Optimization

The goal of optimization is to find a solution to some problem which minimizes (or maximizes) some measure of "goodness." Formally, given a *fitness* function $f: X \rightarrow \mathbf{R}$ over some closed domain X, the goal is to find a value $x \in X$ which minimizes (or maximizes) f. Such a value of x is called a *global optimum* since it is the optimal value over the entire search space. For many problems of interest it is believed that there is no algorithm which is guaranteed to find a global optimum in polynomial time. In such cases, a heuristic is often used to hopefully find a "good" solution with high probability. Common global optimization heuristics include simulated annealing, tabu search, and evolutionary algorithms. This dissertation primarily examines the last of these, and relates it to simulated annealing and the recent Go-With-the-Winners algorithm.

I.A.1 Evolutionary Algorithms

Evolutionary algorithms (EAs) encompass a broad class of heuristic techniques, including genetic algorithms, evolution strategies, evolutionary programming, and genetic programming. What these methods have in common is their use of population-based sampling with successive populations biased towards regions where fit solutions have previously been found. A population of solutions is maintained, initially composed of randomly generated individuals. From this population, solutions are *selected* for *reproduction*. Solutions which are more fit ($f(x)$ is lower) are more likely to be selected, so that the search tends towards regions of the space where good solutions have been observed previously. The reproduction includes some form of *perturbation* of the solution by a set of genetic operators, possibly a recombination (or *crossover*) with another selected individual. Finally, the new solution is placed into the population, usually replacing a current solution.

There is substantial literature devoted to each subclass of evolutionary algorithm, and each has many variants [22, 6]. Figure I.1 outlines the generic algorithm; we briefly describe each of the major subclasses below.

```
Randomly generate an initial population of size M
Repeat
    Evaluate each genotype in the population
    Select genotypes for reproduction, based on fitness
    Perform recombination on selected genotypes
    Perform mutation on selected genotypes
    Replace part or all of population with new genotypes
Until good enough solution found, or out of time
```

Figure I.1: Generic Evolutionary Algorithm

Evolutionary programming (EP) [23, 21] originated as a method to evolve finite state machines to perform various tasks, though its use is now significantly broader. What distinguishes EP from GAs most starkly is the lack of recombination: the population evolves through selection and mutation only. Practitioners of EP argue that for real-world problems, there usually are not composable building blocks for crossover to exploit. The various parameters of a given problem are too interrelated for there to be independently solvable subproblems. We explore this issue in Sections IV.C.2 and IV.D.4.

The evolution strategies (ES) [73, 5] are characterized primarily by their use of *adaptive* mutation sizes. Mutations are typically drawn from some distribution whose variance is itself encoded on the genotype. Hence, the size and shape of the mutations evolve along with the solutions. Also distinguishing ES is the common use of the *steady-state* replacement method, in which a single solution at a time is generated and placed into the population, as opposed to replacing the entire population at once (the *generational* approach).

Genetic algorithms (GAs) [41, 27, 17] are characterized by the importance they place on recombination (or crossover) of solutions. In a well-designed GA, crossover is able to combine successful "subsolutions" from disparate parents to form new solutions with the best of both. In other words, once a subsolution (known as a *building block*) is discovered by any member of the population, it can be propagated to the rest of the population without needing to be rediscovered. This is considered to be the primary source of novel solutions in a GA, though small mutations are usually used as well. Hence, designers of genetic algorithms are careful to design the problem representation and crossover operator to work well together. Being in direct contrast to the approach taken by EP, this is the source of some controversy in the EA community and literature [46].

Historically, GAs have generally used binary representations even for real-valued parameters, although this practice is not as universal as it once was. The genetic algorithm is perhaps the most commonly used type of EA, and it is the type used throughout this dissertation.

Finally, the genetic programming (GP) [54] class of algorithms is concerned with the evolution of programs. Usually the representation is in terms of LISP-type S-expressions, which are essentially trees having operators at all internal nodes and operands at the leaves. Specialized types of mutation and crossover are used to handle this tree data structure.

Though EAs can generally be classified according to the above types, there are no absolute boundaries between them. Many EAs incorporate features of different classes; for example, "GAs" which are steady-state or which do not use crossover are not uncommon. In fact, genetic programming is often considered to be a proper subset of genetic algorithms. We make no claims about one method being generally superior to another, and will use the term EA unless the need arises to be more specific. Most experiments in this dissertation use crossover and constant mutation sizes, and so fall naturally into the GA class. We usually use a steady-state version, however, and occasionally do without crossover. The details of our variant are described in Section IV.D.

I.A.2 Simulated Annealing

Simulated annealing [52, 15] is a different type of global search technique. Instead of a population, a single solution is maintained. At each time step, a small perturbation is made to the current solution, and the resulting new solution is compared to the current one. The new solution replaces the current solution if it is better, or with probability $e^{\Delta_{fit}/Temp}$ if it is worse. Here, Δ_{fit} is the difference in fitness between the two solutions, and *Temp* is the current *temperature*. Therefore, the probability of moving to the new solution decreases exponentially as its fitness gets worse, and also as the temperature gets lower. Usually the temperature is gradually decreased throughout the run, so that uphill moves (for minimization) become less and less likely as the run progresses. Our implementation of SA, which is fairly typical, is described in more detail in Section II.A.4 and Figure II.4.

Initially, SA explores globally, spending more time in regions which on average have fitter solutions. In later stages the search is confined to a small area, and SA simply optimizes within that area. In these final stages it is very similar to local search, as described in Section I.B.

I.A.3 Go-With-the-Winners

Go-With-the-Winners (GWW) [3] (and a variant by Dimitriou and Impagliazzo [19, 20]) is a fairly recent search algorithm that attempts to continually maintain a uniform distribution over all solutions which are better than some explicit *threshold* *fitness.* The algorithm is outlined in Figure I.2. As in EAs, a population of solutions (or "particles") is maintained. The algorithm proceeds in distinct stages, analogous to an EA's generations. Unlike the EA, however, all members of the population are required to be better than the threshold fitness, which gets incrementally more restrictive as the search progresses. At each step, the particles are redistributed according to an estimate of their potential for future improvement. This is similar to selection and reproduction in an EA, although it is not based on fitness and there is no recombination. Finally, each member of the population does a random walk of small mutations (starting from its current position), restricted to solutions whose fitness is within one of the current threshold.

Crucial to understanding this method is the notion of the *search graph.* The nodes of the search graph are simply the possible solutions to the problem. Two nodes are connected with an edge if the corresponding solutions are one "mutation" apart, where a mutation is typically a single bit flip. In GWW, parts of the search graphs corresponding to solutions which are worse than the current threshold are not visited. As the threshold is decreased, therefore, more and more of the graph becomes off-limits. The accessible portion of the search graph eventually becomes disconnected, resulting in separate components which cannot be reached from each other without traveling through the forbidden portion. Through its redistribution and randomization steps, GWW attempts to maintain at least one particle in each component at all times, while maintaining a uniform distribution of solutions within each component. The success of these goals depends on the number of particles used, the length of the random walks (both of which are algorithmic parameters), and the characteristics of the problem instance.

With its focus on uniformly covering the entire search graph, GWW exemplifies *exploration*, as opposed to *exploitation.* Roughly, an exploitative algorithm is one which takes advantage of any quick improvements it finds, possibly at the expense of larger improvements which could be found by a more careful global search. In contrast, an explorative algorithm may forgo easy improvements in the interest

- Initialization Stage: Generate *Pop* random solutions. Set i to be a maximum possible value for a solution.
- Until all particles are at local minima do:
- Stage i:
  - *Pre-redistribution:* Define the *up degree* of a particle as the number of neighbors with value at most $i+1$. Redistribute each particle to one of the current solutions inversely proportional to the up degree.
  - *Post-redistribution:* Define the *down degree* of a particle as the number of neighbors with value at most i. Redistribute each particle to one of the current solutions proportional to the down degree.
  - *Randomization:* For each particle, perform a $2w+1$ step random walk, restricting odd steps to neighbors with value at most i, and even steps to neighbors with value at most $i-1$.
- Go to stage $i-1$.

Figure I.2: Go-With-the-Winners Algorithm for minimization. The constants *Pop* and w are algorithmic parameters. Figure adapted with permission from [14].

of doing a complete search. For an EA, the degree to which selection favors fitter individuals (the *selection pressure*) determines the tradeoff between exploration and exploitation. Note that local search is the ultimate exploitation algorithm, always moving downhill.

While GWW can be used as an optimization algorithm, it has a unique strength in allowing the *analysis of specific problems.* If the number of particles and the random walk length are sufficiently large, the algorithm can gather statistics such as the number of connected components there are at each threshold. Such data can be useful in tailoring specific optimization algorithms to specific applications of interest. This process is described for graph bisection of planted bisection graphs by Carson and Impagliazzo in [14]. Due to the knowledge gained about these graphs from GWW, planted bisection is one of the classes of test problems for our experiments.

I.B Local Search

In contrast to global optimization, *local optimization*—hereafter referred to as *local search* (LS)—is concerned with finding a solution which is as good as or better than all other solutions in its local "neighborhood." Such a solution is called a *local optimum.* At the most basic level, all local search algorithms follow the same algorithm: an initial solution is generated and is repeatedly improved by making small changes. This continues until there are no neighbors which are better. Local search is often easier than global search in that local optima can be found efficiently. Furthermore, a solution can be verified as a local optimum quickly, whereas verifying a global optimum may require examining the entire search space. Though local optima are generally more fit than random solutions, they may be substantially worse than global optima.

There are substantial differences between LS methods for continuous domains and those for combinatorial domains. Continuous LS algorithms make use of explicit *step sizes* and *directions* in the search space. The search progresses by making steps of various sizes in one direction after another until a local minimum is reached. Often, these methods make use of gradient information (either computed directly or estimated by a finite-difference method) to determine the direction of the next move.

Combinatorial problems, on the other hand, have no notion of gradient or even direction in the search space. If a particular move improves the solution, there is no way to "move further" in that direction. Combinatorial LS algorithms generally differ in how the next move is chosen, and in how the neighborhood is defined. The neighborhood of a solution x is defined to be the set of solutions which can be reached from x by a single "step" of the local search algorithm. Hence, different LS algorithms may search over different neighborhood structures, and what qualifies as a local optimum under one algorithm may not qualify in another.

Section II.A.2 reviews the techniques used for both continuous and combinatorial LS, and discusses the differences. We discuss various issues related to combinatorial LS in Section II.A.2 and examine specific algorithms for graph bisection in Section IV.B.

I.C Global-Local Hybrid Algorithms

Global-Local hybrid algorithms combine aspects of both global and local search in order to improve the quality of the final solutions found and the efficiency of the algorithm. Global search techniques need to explore broad regions of the search space and determine where to focus further effort. In general they are less efficient than local search at finding local optima. Hence, a natural synthesis is to rely on a global search method to choose solutions in promising regions of the domain, and then use LS to refine these solutions to local optima.

I.C.1 Evolutionary Algorithm with Local Search

For evolutionary algorithms, the composition of global and local search can be done explicitly: *solutions in the population may undergo LS before being evaluated.*

Note that the global and local aspects of the search inform each other; the EA provides starting points in good regions of the search space, and LS returns information about local optima which may be used by the EA to bias future sampling. We refer to such an algorithm as an *evolutionary algorithm with local search* (EA+LS). Previous work by several authors (for example [62, 1, 38, 71]) has shown that the EA+LS can be a substantial improvement over the EA without LS on a wide variety of problems. In particular, Hart [38] examines several possible ways of using LS in an EA context for continuous optimization, including applying LS to only a fraction of the population, using only a fixed amount of LS per solution, and biasing the selection of solutions which will undergo LS. In this dissertation we extend Hart's analysis to the case of combinatorial problems, examining many of the same issues (including length of LS to use and how to allocate it) and several new issues (including the use of crossover and the ratio of global to local search effort) in this context. Some of the lessons we draw may have applications back to the continuous case.

I.C.2 Simulated Annealing

Simulated annealing combines global and local search in a less explicit manner. When the temperature is high, it is able to explore many regions of the search space, while the lower probability of accepting inferior moves biases it towards spending somewhat more time in regions where it finds better solutions. As the temperature is lowered, however, the search becomes increasingly localized. Eventually, when the temperature is very low, the only moves which are accepted are improving moves. At this point the search is equivalent to a local search. Hence, SA transitions from global to local search in a fairly continuous manner.

I.C.3 Go-With-the-Winners

The Go-With-the-Winners algorithm also blends aspects of global and local search. By maintaining a population and attempting to keep it uniformly distributed among the allowable solutions, it covers the overall space as well as an EA. On the other hand, by relentlessly forcing the entire population to have better fitness than the threshold, it is similar to local search, in which each move improves the solution.

I.D Dissertation Overview

Techniques described in this chapter are examined in detail in the dissertation. Experiments examine the EA, local search from random starting points (so-called *Monte Carlo local search*), simulated annealing, and especially the EA+LS hybrid.

Chapter II provides background information for local search, evolutionary algorithms (with and without local search), and simulated annealing. The difference between continuous and combinatorial optimization is explored in Section II.A.2.Some outstanding issues relating to EA+LS are discussed, including the allocation of local search to members of the population and the methods of applying LS results.

Chapter III lays out our intuitions about how an EA+LS search progresses, especially with regard to the effects that LS has on the global search. Also considered are the ways in which the presence of LS modifies the role of the usual genetic operators (recombination and mutation). These intuitions lead to several expectations and recommendations, which are tested in the experiments of Chapter IV. Finally, some related complexity theory results are examined, especially the Polynomial-Time Local Search complexity class.

Chapter IV presents the results of a series of experiments which explore all of the above themes. The experiments are carried out for graph bisection on instances from a variety of classes. Graph characteristics that are likely to affect search (number of local minima, basin sizes, and stability of local minima) are explored for each class. Baseline results are presented for Monte Carlo local search under various local search methods, with a focus on the tradeoff between solution quality and search time. The EA without local search is evaluated to examine the role of the genetic operators when LS in *not* used. Then, several EA+LS parameters are explored, including the mutation size, use of crossover, local/global search ratio, LS length, and whether the partitions resulting from local search are used by the EA. We analyze the results in terms of our expectations, as presented in Chapter III. We conclude with a performance comparison of EA+LS and simulated annealing.

Chapter V presents a discussion of the major findings, open questions, and directions for future research.

Chapter II

Background

II.A Prior Knowledge

II.A.1 Evolutionary Algorithms

Algorithm Description

Evolutionary algorithms (EAs) have become very popular as a method for "global" search or optimization. They do global sampling by maintaining a population of genotypes, initially uniformly distributed throughout the search space. Succeeding generations are produced from previous ones by means of selection (biasing the population towards areas where the current generation shows greater promise) and genetic operators, such as recombination and mutation.

More specifically, feasible solutions to the optimization problem are represented as a set of parameters, known as a genotype. The parameters may be real-valued or discrete, depending on whether the problem being solved is continuous or discrete. As an example, an evolutionary algorithm for the traveling salesman (TSP) problem might represent a solution as a list of integers specifying the nodes in the order in which they are visited. Initially, the EA randomly generates many genotypes, and these are taken to be the initial population.

During each generation, every genotype in the population is evaluated and assigned a fitness value, based on how good a solution it is. Genotypes are selected to reproduce based on how good their fitness is. Better genotypes get to generate more offspring, and therefore their genes have a greater prevalence in the next generation. In this way succeeding generations are biased towards more promising regions of the search space based on the statistics that are collected earlier on.

A new generation is produced by applying the genetic operators to on the selected genotypes from the previous population. In the case of the genetic algorithm, this is done by performing recombination, or crossover, Typically this involves taking two of the selected "parents", copying some of the parameters from one parent, and copying the rest of the parameters from the other parent. Recombination of selected genotypes is the primary means of generating new solutions. Mutations, small random changes, are often done to the resulting genotypes. These are generally considered to be helpful, if done in small enough quantities. An outline of the entire process is given in Figure II.1.

```
Randomly generate an initial population of size M
Repeat
     Evaluate each genotype in the population
     Select genotypes for reproduction, based on fitness
     Perform recombination and mutation on selected genotypes
     Replace part or all of population with new genotypes
Until good enough solution found, or out of time
```

Figure II.1: Standard Evolutionary Algorithm

Generational vs. Steady-State

The description above left open the question of how many new individuals are generated at once, and how the individuals to be replaced are chosen. Most commonly, the entire population is replaced at once, so that P new individuals are generated in each generation, where P is the size of the population. An EA using this scheme is called a *generational* EA, to emphasize the fact that the population is modified in distinct generations. A frequently used variant on the generational EA is to keep the single best individual in the population for the next generation, so that the search never loses the best solution seen. This technique is known as *simple elitism.*

In contrast to the generational method, a *steady-state* EA replaces only one individual at a time. The individual to be replaced is usually the worst individual in the population. A steady-state EA may be more stable, as the best solutions do not get replaced until the newly generated solutions become superior. Though it is less common than the generational technique, we will use steady-state EAs for most of our experiments, as it allows the results of LS to be maintained in the population. We discuss this issue further in Section II.B.1.

II.A.2 Local Search

Generally stated, the goal of local search is to find a solution which is as good as or better than all of its surrounding points. The meaning of this depends on the problem at hand: for continuous problems we define local minima in terms of open subsets of R^n, while problems on a discrete domain employ a specified neighborhood structure to define local optima. We briefly describe the major approaches for both the continuous and combinatorial cases, and then discuss some of the fundamental issues which make the cases different.

Continuous

Local search techniques on continuous domains can generally be described as falling into two classes, *direct* and *indirect* search, based on what information is available to the local searcher. Direct methods use no information other than the fitness values of the points they sample; in particular, no use is made of gradients or second derivatives or approximations to them. Common direct search techniques include the downhill simplex method due to Nelder and Mead [63, 68] and the Solis-Wets algorithm (SW) [78]. The latter attempts moves of the current "step size" along all axes until an improvement is found, with conditions for shrinking or expanding the step size in response to the frequency with which improving moves are found. Pattern search methods [82, 18, 81] have also come into frequent use.

Indirect search methods are often used when gradient information is efficiently computable. The simplest such method, gradient descent, repeatedly computes the gradient at the current location and moves in the opposite (for minimization) direction. Different algorithms differ in how much the step size is changed and under what conditions. More sophisticated *conjugate gradient* methods use gradient information in an attempt to deduce the direction of the local minimum. At each stage of the algorithm a line-search minimization is done along the current direction. For quadratic functions in n dimensions, the process converges on the minimum after n such line-searches. Finally, the *quasi-Newton* methods attempt to estimate the Hessian of the function through repeated line-minimizations. For a more complete discussion of the various methods see [68].

Combinatorial

Local search methods on combinatorial domains all follow the same basic pattern: choose an initial point, then repeatedly move to a neighbor which is better than the current point. While this is simple enough, there are many details regarding how the improving moves are chosen and what qualifies as a "neighbor" which differentiate various LS algorithms. Note that LS for combinatorial problems is always a direct search method, since there is not a notion of gradient. Figure II.2 gives the canonical discrete LS algorithm. The initial step (generate an initial solution) is generally done uniformly at random over the search space. Alternatively, in an EA+LS hybrid, the evolutionary algorithm is used to generate the initial starting points, as discussed in Section II.A.3.

```
Generate an initial solution $P$
Repeat
    Move to a neighbor $P'$ with better cost
Until $P$ has no neighbors better than itself
```

Figure II.2: Canonical Local Search Algorithm

Neighborhood Structure Local search methods assume a *neighborhood structure*, which defines what moves are legal. Formally, a neighborhood structure N on a search domain S is a function $N: S \to 2^S$ which specifies the set of neighbors for each point in the search space. For local search to be feasible the size of each point's neighborhood must be small compared to the size of the search space. For problems which can be represented as n-bit strings, for example, a typical neighborhood for a point is just the collection of strings which are a single bit-flip away; such a neighborhood has size n. Additionally, the neighborhood structure is usually symmetric (i.e. if $y \in N(x)$ then $x \in N(y)$), but this it not a requirement. In fact the well-known Kernighan-Lin LS method for graph partitioning [51] employs a nonsymmetric neighborhood structure.

The success of the local search algorithm will depend on the neighborhood structure chosen. There are no general rules for how to choose neighborhoods, but in general larger neighborhoods may be expected to result in better local minima. To see this note that any local minimum under a given neighborhood structure N will also be a local minimum under any structure N' for which $N'(x) \subset N(x)$ for all x. The average quality of the local minima found under a given neighborhood structure is referred to as the structure's *strength*. Strong neighborhoods result in better solutions, though the time required to find the solutions may also be greater.

Move Decision The second decision that must be made in defining a LS algorithm is the choice of move decision, i.e. among the neighbors which are better than the current solution, how does the algorithm choose which one to move to? There are two standard decision rules, *steepest descent* and *first-improve.* The steepest descent method always chooses the very best neighbor. Note that this requires the entire neighborhood to be searched at each step.

The first-improve method simply moves to the first neighbor it examines which is better. Hence, the algorithm may examine only a fraction of the neighborhood of each point it visits. For this reason first-improve is often faster than steepest descent, though it may also result in different local minima. An additional issue with first-improve is the order in which the neighbors are examined. Since it moves to the first improvement seen, different orderings may result in different local minima. Generally the neighbors are examined in a random order. But it may be advantageous to used a fixed ordering if by so doing one can take advantage of knowledge about the problem being searched. For a broader discussion on combinatorial LS issues see [66].

Differences between Continuous and Combinatorial

There are many significant differences between continuous optimization and combinatorial optimization which affect how we do EA+LS. One difference has to do with the nature of global search. Combinatorial problems of interest are generally NP-complete, whereas continuous problems are usually not analyzed in terms of complexity. Being difficult to solve, an NP-complete problem may be expected to have higher-order dependencies between genes. This makes it harder for the EA to perform an effective search. Alternatively, it requires more intelligent design of the genetic representation and operators.

Aside from global concerns, the notion of local search is completely changed in the combinatorial domain. Continuous local searchers make explicit and repeated use of the notion of *direction.* Typically moves are made in a given direction until no more progress can be made. Then a new direction is chosen. The moves being made may be of varying sizes, depending on the success of previous steps.

In contrast, combinatorial problems do not have the same notion of direction. Swapping a node from one side of a partition to the other or changing a variable from True to False are all-or-nothing events which cannot be extended. A successful LS step cannot be capitalized upon by doing more of the same. An immediate consequence of this is that there is no notion of gradient. Gradients (analytic or approximated) are commonly used in continuous optimization in so-called *indirect search.* All search in the combinatorial case must be *direct*, relying only on functions cost directly discovered during the search. Standard direct search techniques for continuous problems include the downhill simplex method due to Nelder and Mead [63, 68] and the Solis-Wets algorithm (SW) [78]. But while both of these operate without a gradient, they still make use of direction, and so no direct analogue exists for combinatorial problems.

Another consequence of there being no direction is that there is not the same concept of *step size.* Hart has shown that issues regarding initial step size and how it can be adapted during search can be critical to the success of an EA+LS for continuous optimization [32, 37]. The only notion of step size in our case is related to how many parameters can be changed in a step (e.g. swapping two nodes at a time instead of one). This does correspond to having neighbors which are varying distances away, but it is not the same. Increasing the step size here also increases (combinatorially) the *number* of neighbors, and hence changes the neighborhood more radically than in the continuous case.

One way in which Hart has found varying step size to be helpful is that large step sizes allow local search to "jump over" smaller basins to find better solutions which would not necessarily be reachable by way of a continuous monotonically improving path. In this way the "local search" can do a larger scale search. This is thought to be helpful mainly early in an EA+LS run, when the global search is still in full swing. Later on, the step sizes are decreased, and LS assumes the role of a refinement operator, zeroing in on local optima. Such a dual role cannot occur in the combinatorial case which has fixed step size. Single step moves will never be able to jump out of basins (This does not mean that every solution is associated with a single basin. Many solutions can reach multiple basins through paths of improving single steps). In a sense, this makes the role of LS in a combinatorial EA easier to understand, as it is restricted to do "pure LS".

A related issue is that of the minimum scale of search. Continuous problems are notable for not having a smallest scale. In practice, it is not usually known a priori what the minimum feature size of the search space is, or how close to optimal is close enough. The limiting factor is often decided by the floating point precision of the machine being used. Combinatorial problems have an absolute minimum scale which is known in advance, namely the minimal move defined by the neighborhood structure. One important consequence of this is that it allows for well-defined stopping criterion for LS.

Combinatorial search problems of interest generally have hundreds or thousands of dimensions. Continuous problems usually do not have this many dimensions, and can be interesting and difficult even with ten or fewer dimensions. Accordingly, exhaustively searching the neighborhood requires more effort in the combinatorial case. More solutions have to be considered per local search step, and this may affect the relative value of LS.

A final difference between continuous and combinatorial problems concerns the existence of well-defined local basins and local optima. Basins are well-defined in the continuous case; specifically, the basin associated with a local optimum is the set of points from which a continuous monotonically improving path could not end up at any other local optimum. A local minimum (maximum) can be defined as a point which has a neighborhood containing only less fit points. Formally, a point $x \in R^n$ is a local minimum if there is an $\epsilon > 0$ such that $f(x') > f(x)$ for all $x' \in R^n$ with $\|x' - x\| < \epsilon$. Note that neither of these definitions refers to any particular local search method. The basin structure is implicit in the space R^n by way of common analytical notions.

Combinatorial problems, in contrast, have basins and local optima only with respect to particular LS methods or neighborhood structures. They are not implicit in the problem itself, and may be very different under different neighborhood structures. Somewhat paradoxically, the ability to find a local optimum and verify its optimality are trivial (conceptually, not necessarily computationally) in the combinatorial case. In the continuous case, exact local optima are generally not attainable (due to precision limits and finite search times), and verifying that a point is a local optimum is not possible without the use of gradient information.

II.A.3 Evolutionary Algorithm Plus Local Search Hybrid

Algorithm Description

An often-used extension to the EA as described above is the addition of a local searcher. A local searcher is an algorithm like gradient descent or random hill-climbing which searches only for a local optimum. These methods are not meant to do any sort of global optimization, leaving this up to the population and genetic operators. The local search algorithm is applied to members of the population after recombination and mutation, and the fitnesses at the local optima are used for the selection step. The pseudo-code for this algorithm is shown in Figure II.3.

```
Randomly generate an initial population of size M
Repeat
    Evaluate each genotype (or locally optimized genotype)
    Select genotypes for reproduction, based on fitness
    Perform recombination and mutation on selected genotypes
    Perform local search on members of the population
Until good enough solution found, or out of time
```

Figure II.3: Evolutionary Algorithm with Local Search

For continuous optimization problems, there are good reasons to expect local search to be a beneficial addition to the evolutionary algorithm. The view we take is that the global population-based search is complemented by a local refinement operator. Essentially, we expect the EA, with its (hopefully) diverse population and recombination operators, to perform a "global" sampling over the entire search space. It gathers statistics about the various regions and then gradually focuses search on regions with better average solutions.

Local search has a complementary role to the global searching of the EA. It is not expected to search a large portion of the search space, as presumably the local search basins of attraction will be small compared to the size of the search space. One immediate potential benefit offered by local search is that it improves the quality of the statistics gathered during the EA's global sampling. The local optima provide a consistent characterization of the various basins (namely, the lowest point). Hence, the EA is likely better served by the values at the local optima than it is by values at random points.

Another reason that a local searcher can help is that EAs are not efficient hill-climbers. On optimization problems that are solvable by local search methods (e.g. gradient descent, conjugate gradient, etc.), it is generally the case that the local search algorithm will find the optimum faster than an EA. This has been shown empirically many times [24, 86, 25] for continuous problems. This is partially because of their exploitation of gradient information. However, in Section IV.C.1, we will see empirical evidence that this is also true of combinatorial problems, for which gradient information cannot be exploited.

Given that we have a good local searcher, the hope is that the EA searches over *basins*, rather than points. Ideally, new individuals generated by the genetic operators would be in different basins of attraction. If there are not too many local basins, then the EA may be able to search all or almost all of them. More likely, there will be many more local basins than members of the populations. In this case, the EA has to rely on there being regularities over the local optima that can be exploited. Of course, if there are very few local basins, then multiple runs of the local searcher from random initial positions may be best, eliminating the need for adaptive global search.

Lamarckian vs. Darwinian Evolution

The algorithm description above does not specify what is done with the results of local search, other than that the resulting fitnesses are used for selection. The resulting solution itself can be dealt with in one of two ways. Commonly, the result of LS replaces the pre-optimized solution in the population, i.e. the results of LS are copied back onto the genotype. This strategy is known as *Lamarckian* evolution, as it allows "acquired traits" to be passed on genetically. For certain problems in which the genotype is not equivalent to the phenotype it may not be possible to use this strategy. Section II.B.2 discusses under what situations it may be appropriate.

The alternative to Lamarckian evolution is *Darwinian* evolution, in which the solution resulting from LS is discarded—only its fitness influences the search. This technique is used when Lamarckianism is not an option, and sometimes even when it is. The benefits of both methods are discussed in Section II.B.2.

Lessons from Continuous

In his Ph.D. dissertation [38], Hart examined the role of local search when used in conjunction with EAs. This work was concerned primarily with continuous optimization problems, including molecular conformation, training of neural networks, and some complicated artificial test problems. The basic finding was that EAs with local search were superior to EAs without local search. On every problem studied, a sophisticated local searcher such as conjugate gradient was found to be beneficial to the EA. This means that the EA with local search was able to find better solutions in the same amount of time.

A common way to use local search in a EA is to apply it to every member of each population. The resulting solutions replace the population members, and are used to generate the next population under selection and recombination (so-called Lamarckian evolution). Hart investigated several variations on this scheme which are generally applicable. The most important variant is the use of a small local search probability. Instead of applying local search to every member of the population, it is only applied to members with some (typically small) fixed probability. This was very often beneficial over always using local search, and often remarkably so. Even more benefit was obtained by allowing the local search probability to be adaptive, changing according the diversity of the population.

Another possibility explored by Hart was the imposition of a maximum search length. Instead of allowing each local search to go to completion, they were cut off after some fixed number of steps. This was usually seen to be beneficial. Finally, a variant mentioned by Hart but not explored was the possibility of "non-Lamarckian" evolution. In this scheme, local search is performed only to obtain a fitness value, but otherwise does not affect the genetic material which is passed on. This has been explored by others [8, 48], but is generally not used for optimization.

It is worth noting that the best results found in Hart's dissertation were for gradient-based local search methods. This is not particularly informative for discrete optimization, since there isn't the notion of a gradient. Hart also examined a direct local search method (using no gradient information) developed by Solis and Wets [78]. This was found to be quite beneficial is some cases but not in others. More recent work by Hart has found that a different kind of direct search, known as pattern search [82, 18, 81], is more effective than Solis-Wets as an addition to the EA [33, 32, 37].

Thin Film Metrology Recently, we have applied an EA to the industrial problem of thin film metrology [56]. Various measurements are made of the physical properties at the surface of a processed semiconductor wafer. The task is to determine the actual structure and materials present in the stack of films on the surface. This is of great importance to chip manufacturers as a verification that their fabrication processes are working correctly.

The physical measurements provide only indirect information about the structure of the film stack; models must be found which match the data in order to infer the structure. The parameters of these models include such things as the thickness, index of refraction, and extinction coefficient of each of the layers. The EA has been used successfully to search over model-space and provide good fits to the data. We have found that adding a sophisticated local search method (a Levenberg-Marquardt type nonlinear least-squares fitter) to the EA can result in significantly better fits, using the same amount of computational effort.

II.A.4 Simulated Annealing

Simulated annealing (SA) [52, 15] provides an especially interesting contrast to an EA+LS, in part because it has proven itself to be an effective, robust method in many applications. It can also be viewed as a global/local hybrid: at high temperatures it explores more or less without restriction, or globally. As the temperature is decreased, fewer and fewer uphill moves are made, and the search is more and more local. EA+LS variants "factor" global and and local aspects into separate but interleaved searches rather than randomly selecting between the two criteria at each step.

The basic algorithm is presented in Figure II.4. SA can be thought of as a generalized version of local search: the algorithm maintains a single solution and repeatedly examines neighbors of this solution. Whenever an improving neighbor is found that neighbor becomes the new current solution. What differentiates SA from local search is what happens when a *worse* neighbor is encountered. SA moves to this neighbor with a probability depending on the difference in fitness between the current solution and the neighbor, and also on the current "temperature." The worse the neighbor is, the less likely SA is to move there. Furthermore, as the run progresses the temperature decreases, which exponentially lowers the probability of moving to a worse neighbor. Eventually, this probability is so low that virtually no moves are made to worse neighbors. At this point SA is equivalent to local search, as all moves are improving.

There is well-regarded study by Johnson et al. of simulated annealing applied to graph bisection [44]. We use this study as a baseline for our EA+LS experiments.

```
Randomly generate a solution, $current$
Repeat
    Repeat for $N$ steps
        Perturb $current$ to get $new$
        If ($new$ is better than $current$), $current$ = $new$
        Else replace $current$ with probability $e^{(f_{new} - f_{current})/Temp}$
    $Temp$ = $Temp * R$
Until no improvement or out of time
```

Figure II.4: Canonical Simulated Annealing Algorithm

Furthermore, where we perform SA experiments (see Section IV.E.1) we use the same algorithmic choices used there. Specifically, we choose an initial temperature which results in approximately 40% of moves being accepted. This temperature is determined empirically prior to the SA run. The temperature reduction factor (R) is 0.95. The maximum number of steps spent at a given temperature (N) is $16n$, where n is the number of nodes in the graph. The temperature is decreased more quickly if the percentages of moves which are accepted is overly high. Specifically, whenever $9n$ moves have been accepted at a give temperature it is decreased. Finally, the run is terminated when the percentage of accepted moves falls below 2.0% for five consecutive temperature values.

A final note about the details of the graph partitioning search space: as discussed in Section IV.A, the search is performed over "unbalanced partition space," in which the partitions considered may not have equal-size subsets. An imbalance penalty is applied, but nevertheless there may be unbalanced partitions which are local minima. Hence, we need a mechanism to "repair" unbalanced partitions at the end of the SA run. The method we use is greedy, bringing the partition into balance one node at a time, where the node chosen at each step is that which results in the smallest increase (or largest decrease) in cost.

II.B Outstanding Issues

II.B.1 Generational vs. Steady-State

The question of whether to use a generational or a steady-state EA is often not given much thought or is regarded as a matter of personal preference. However, there may be solid reasons to favor one over the other in most cases. As a general observation note that doing selection, creation, and evaluation one solution at a time permits the information obtained with each new solution to be used in the creation of the next. Contrast this with the generational approach in which an entire population of solutions is generated at once, without regard for the results of preceding new solutions. An algorithm with the freedom to use preceding results to bias successive solutions would have to be able to do at least as well as this; in the worst case it could just ignore all preceding solutions in the current generation. Direct comparisons between generational and steady-state algorithms in the literature are sparse, but see [29, 80, 26, 85, 47].

II.B.2 Lamarckian vs. Darwinian

A central issue in an EA+LS is how to use the results of LS. An algorithm drawing inspiration from biological evolution might use the new fitness value for purposes of selection, but would not carry the new solution itself into the next generation. In other words, local search affects selection, but does not modify the genotype. This so-called Darwinian evolution respects the notion that changes which occur during a lifetime (analogous to local search in our case) cannot be genetically propagated to future generations.

As algorithmic engineers, we often are not bound by this restriction. In many EAs with LS, the new locally optimized solution replaces both the genotype and the fitness of the old solution in the population. This so-called *Lamarckian* evolution does not occur in nature because of the difficulty of reverse transcription of phenotypic traits back onto the genotype.[1] For directly encoded optimization tasks, this does not usually present a problem. Local search acts on the same representation as the genotype, and so the "reverse transcription" in this case is just copying the bitstring representation.

The question of whether to be Lamarckian or Darwinian in practice has generally received little attention in the literature. Sometimes the Darwinian choice is forced; this might occur if the genotypic and phenotypic spaces are substantially different. For example, there may be a *developmental* process during which a genotype is "decoded" into a phenotype. The inverse of this mapping may be impractical or impossible to compute, and hence changes which occur to the phenotype before reproduction cannot be mapped back to the genotype. For a general discussion of development and Lamarckian issues in evolutionary algorithms see [36] and [35]. Examples of such developmental systems include developmental neural networks [30], grammar-based sorting networks [50], stochastic grammars [49], cellular automata rules [55], and virtual block creatures [76].

Furthermore, it may be the case that LS produces solutions for which there is no genetic encoding. As an example, in a protein folding application, Hart used a genetic representation which coded for discrete bond angles (60^o increments), whereas local search was performed over the real domain [38, p. 107]. Another example is that of evolving neural network weights as starting points for backpropagation [8]. The range of weights which could be genetically encoded was only a restricted subset of the weights which backpropagation could lead to. The genetic representation in both of these examples is not expressive enough to encode the results of local search, and so Darwinian evolution has to be used.

Despite the above concerns, Lamarckianism is often an option for optimization applications. This is because current practice is usually to have a genetic representation which directly encodes solutions (no development, so genotype equals

[1]But note that in recent years it has been observed that some simple bacteria seem to be able to *direct* mutations in response to the environment [11]. Though not quite Lamarckian (the bacteria do not individually adapt *themselves* to the environment, only their offspring), it is one way in which the strict Darwinian view is violated.

phenotype), and which allows all possible solutions to be encoded. There is no obstacle to using Lamarckian evolution, and the issue is simply whether this leads to better performance than Darwinian evolution. In such a case Lamarckian evolution is generally used in practice. It makes more intuitive sense, as the Darwinian alternative constantly throws out much of the results of the expensive local searches. Also, Lamarckianism has been seen empirically to do better on some instances of both continuous problems [38, 56] and combinatorial problems [2]. However, Whitley et al. [84] have shown that there do exist discrete optimization instances for which Darwinian evolution is more likely to find the global optimum, though it is much slower to converge.

Go-With-the-Winners Darwinian?

Another viewpoint on this issue can be had by considering the Go-With-the-Winners algorithm of Aldous and Vazirani [3]. In particular, a variant of this by Carson and Impagliazzo [14] is illustrative. In this search algorithm, a population of solutions is maintained. At any point, all members of the population are better than some threshold fitness, which gets more restrictive as the search progresses (see Section I.A.3). At each step, each solution does a random walk on the search graph, restricted by the current threshold. The length of this random walk is sufficient that the solutions in each component of the search graph are effectively uniformly distributed within that component. This uniformity is critical for finding the global optimum. The relative "fitnesses" of the solutions don't matter as long as they are spread out evenly. This is more similar to Darwinian evolution than Lamarckian. If we make the analogy that a local basin is like a connected component, with local search playing the role of the random walk, then Darwinian evolution is concerned with keeping solutions which are "connected to" (in the same basin as) good solutions, but does not keep track of the good solutions themselves.

II.B.3 Local Search Selection

A key feature of the use of LS in an EA is that it need not be applied to every solution in the population. In many cases, applying LS to as little of 5% of each population results in faster convergence to good solutions. This somewhat surprising effect was first observed by Hart [38, Chapter V], and has since been confirmed by others in different contexts [71]. See Section II.A.3 for more details.

In light of this observation, the question naturally arises as to how best to select the solutions which will undergo LS. Typically solutions are chosen uniformly at random from the population, but this is not the only possibility. This issue of *local search selection* is distinct from reproduction selection, though they are similar in form: both involve choosing a subset of the population. The difference lies in what this subset will be used for, and therefore how it should be chosen.

An obvious method for LS selection is to use the same procedure that reproductive selection uses, namely biasing towards the selection of 'fit' solutions. This approach has been used for the MAX-SAT problem by Grundy [31] and for continuous optimization by Hart [38, Chapter V], though it is usually not used for fear of losing diversity too quickly. Another approach, investigated by Hart in [38, Chapter V] is to choose diverse solutions, hoping to use LS on a representative sample of the population and to avoid premature convergence. See Section III.C.3 for a more thorough discussion of these methods.

II.B.4 Simulated Annealing vs. EA+LS

Given the widespread use of the simulated annealing algorithm, and the fact that it has both global and local search behaviors, it is natural to compare it with EA+LS. What problems are best handled by SA and what problems are best handled by EA+LS? Unfortunately, it is very difficult to make informative empirical comparisons between the two algorithms. Each requires a fair amount of practitioner skill (for example setting the genetic operator probabilities for EA+LS, or choosing an annealing schedule for SA), and the two sets of practitioners are largely disjoint. An implementation of one of the algorithms by a non-expert will always be suspect from the point of view of an expert.

Nonetheless, empirical comparisons have been made ([16, 42, 83]) with mixed results. One of the aims of this dissertation is to make a strong, detailed comparison. A well-known and respected empirical study of SA on graph bipartitioning ([44]) is taken to be an "expertly done" SA baseline. The same problem instances are attacked by the EA+LS here; see Section IV.A for details.

Theoretical comparisons are also difficult to come by. One result by Hart [34] shows, given some conditions, that EAs have a higher probability of finding a global optimum if both algorithms are run for long enough (measured by number of potential solutions considered). However, "long enough" may be impractically long, perhaps longer than the the time required to search the entire search space.

It is possible to gain some insight by considering what is known about SA. Sorkin has shown [79] that SA can be expected to work well roughly when the height of the barrier between any two adjacent basins is proportional to the difference in fitness between the basins' optima. This ensures that the temperature scale is always appropriate. For example, two local optima whose fitnesses are nearly equal cannot be effectively distinguished by SA until the temperature gets low. But at low temperatures SA is unable to climb out of large basins. So in order to compare the two optima, the barrier between them must be correspondingly low. Note that the EA+LS does not suffer from this problem: if the global operators (recombination and mutation) are working as expected, they will make changes large enough to escape from local basins. This suggests at least one type of problem where EA+LS might be expected to outperform SA.

Chapter III

Intuitions and Arguments

In this chapter we present our view of several issues relevant the EA+LS hybrids. Section III.A makes a connection to the some complexity theory results, including the *Polynomial-Time Local Search* complexity class. In Section III.B we lay out some assumptions about how search progresses in an EA+LS algorithm. Section III.C discusses how the role of standard EA operators changes in the context of local search, and how these operators may be modify to complement LS.

III.A LS Complexity Theory Arguments

One advantage of working on combinatorial problems is that we can potentially exploit knowledge gained from complexity theory. General results regarding local search (specifically work on the Polynomial-Time Local Search complexity class) imply that local search itself can be intractable. This obviously raises concerns about any global search algorithm which applies LS multiple times. We will review the theory below, and argue that it is not directly relevant for our purposes. Furthermore, for many problems of interest (including restricted versions of graph partitioning, TSP, and MAX-SAT) a simple complexity argument provides justification for the use of a heuristic method such as an EA to provide starting points for LS.

Consider any optimization algorithm which operates by applying a LS subroutine to a succession (or set) of generated starting points. To simplify this argument assume that the LS method is deterministic, so that local basins are well-defined, i.e. every solution in the search space corresponds to exactly one local optimum under the given LS method.

How quickly can a starting point be found from which LS will produce an optimal solution? Put another way, how hard is it to identify any single point in a local basin corresponding to a global optimum? The answer obviously depends on the characteristics of the original problem, call it L. In some cases this question is easy to answer. If the original problem L can be solved in polynomial time, for example, then such a starting point can be found in polynomial time. The global optimum will suffice.

More difficult to analyze is the case where L is NP-complete. If we assume for the moment that any given execution of the LS method is guaranteed to complete in polynomial time, then no polynomial time algorithm can find an optimal starting point, unless $P = NP$; if there were such an algorithm we could solve L in polynomial time. So with a polynomial time LS method for an NP-complete problem, *any practical algorithm for choosing starting points must be heuristic.*

III.A.1 Cases where Local Search is Easy

Note that the number of improving steps made by a LS is bounded by the number of distinct solution costs. If there are at most a polynomial number of such costs, and if the size of any neighborhood and the time to compute a solution cost are also polynomially bounded, then the LS algorithm will complete in polynomial time.

In particular, binary graph bipartitioning, an NP-complete problem and a major test domain for this dissertation, completes local search in polynomial time when using polynomial size neighborhoods. This follows from the fact that the minimum cost of a partition is 0 and the maximum cost is $n^2/4$. To see this note that there are $n/2$ nodes in each set of the partition. If every node in each set is connected to every node in the other set (the worst case) then the partition cuts $(n/2)(n/2)$ edges. Furthermore, if LS explores *unbalanced* partitions (as some common neighborhoods for graph partitioning do), the maximum partition cost is no larger: if the two sets of the partition have $n/2-k$ and $n/2+k$ nodes, and if every node in each set is connected to every node in the other set, then the partition cuts $(n/2-k)(n/2+k) = n^2/4-k^2$ edges. Finally, note that computing the cost of a solution takes polynomial time, and we have:

Lemma 1 *Any local search algorithm employing polynomial size neighborhoods on the binary graph bipartitioning problem completes in polynomial time.*

Note that the standard neighborhoods for bipartitioning, including all neighborhoods examined in this dissertation, are polynomial size. These include swapping any two nodes, moving a single node across the partition (allowing unbalanced partitions), and the somewhat more complicated neighborhood associated with the Kernighan-Lin algorithm [51] (and see [45]).

This is not all we can say about problems for which local search is easy. Other examples in which the number of distinct solution costs is polynomially bounded (and hence for which local search takes at most polynomial time) include weighted integer graph partitioning where the edge weights are bounded by a polynomial in the instance size, TSP with integer edge weights polynomially bounded, and MAX-kSAT. The proofs for each of these are simple, and are given in turn below.

Define *integer poly-weighted graph k-partitioning* (IPWGP$_k$) to be the problem of graph k-partitioning (dividing a graph into k equal size subsets so as to minimize the total weight of the edges between subsets) restricted to classes of graphs which have nonnegative integer edge weights which are bounded from above by some polynomial in the number of nodes. Note that this includes the binary graph bipartitioning problem discussed above, and hence is still NP-complete.

Lemma 2 *Any local search algorithm employing polynomial size neighborhoods on* IPWGP$_k$ *completes in polynomial time.*

Proof:

It suffices to show that the number of distinct solution costs is polynomially bounded. Since all edge weights are nonnegative integers, the cost of any partition will be a nonnegative integer, and hence we need only show that the maximum possible partition cost is polynomially bounded.

Let n be the number of nodes in the graph. For a given k-partition, each of the subsets contains n/k nodes. Hence the number of edges from nodes in a particular subset to nodes not in that subset is $(n/k)(n - n/k)$. Multiplying by k to account for the edges from each subset, and dividing by 2 to correct for duplicate counting of edges, there are a total of $(k/2)(n/k)(n - n/k) = n^2(1 - 1/k)/2$ edges between subsets. Since each particular weight is bounded by a polynomial $p(n)$, the total cost of any partition is $O(n^2 p(n))$. $\square$

Similar to the above, define *integer poly-weighted TSP* (IPWTSP) to be TSP restricted to classes of graphs which have nonnegative integer edge weights which are bounded from above by some polynomial in the number of nodes. Note that this problem is still NP-complete for non-trivial polynomials since TSP is strongly NP-complete.

Lemma 3 *Any local search algorithm employing polynomial size neighborhoods on* IPWTSP *integer poly-weighted TSP completes in polynomial time.*

Proof: As before, it suffices to show that the maximum possible tour cost is polynomially bounded. Let n be the number of nodes in the graph. Then the cost of any tour is the sum of n edge weights. Since each weight is bounded by a polynomial $p(n)$, the cost of a tour is $O(np(n))$. $\square$

Lemma 4 *Any local search algorithm employing polynomial size neighborhoods on MAX-kSAT completes in polynomial time.*

Proof: It suffices to show that the number of distinct solution costs is polynomially bounded. But given that the solution cost is the number of clauses satisfied, this follows immediately from the fact that there are $O(n^k)$ possible k-clauses over n variables. □

Lemmas 2-4 are not meant to be a complete characterization of the problems for which local search is easy. Rather they are quick results about some of the most common problems in combinatorial optimization.

III.A.2 Polynomial-Time Local Search Complexity Class

Despite the above results for various combinatorial optimization problems, local search methods cannot always be guaranteed to run in polynomial time. In fact, the subject of the computational complexity of local search has received much attention in the literature since it is a general approach which has often been quite effective empirically. The basic theoretical work involves the complexity class *Polynomial-Time Local Search* (PLS) and is described by Johnson, Papadimitriou and Yannakakis in [45].

In this paper the class PLS is defined to be the class of search problems associated with finding a local optimum for a given problem and neighborhood structure. More formally, a problem L is a PLS-problem if the following conditions hold:

- Each instance x of L has a finite number of solutions of polynomial length.
- There exists a *cost* function $c(s, x)$ returning a nonnegative integer for each solution s of x.
- There is a *neighborhood* function $N(s, x)$ returning a set of solutions to x (called the neighborhood of s)
- Finally, there exist three polynomial time algorithms A, B, and C, such that:
    1. Given an instance x of L, $A(x)$ returns some solution s of x.
    2. Given an instance x and a solution s, $B(s, x)$ determines if s is a solution of x, and if so returns the cost of s, $c(s, x)$.

3. Given an instance x and a solution s, $C(s, x)$ returns another solution $s' \in N(s, x)$ with a better cost (i.e. $c(s', x) < c(s, x)$ in the case of minimization) if such a solution exists. Otherwise it reports that no such solution exists.

Roughly this definition includes any discrete problem with nonnegative integer costs for which standard "nice" assumptions hold (polynomial size representations and neighborhoods, polynomial time cost function, finite search space). In particular, integer graph partitioning, integer TSP, and MAX-SAT are included given suitable neighborhood structures.

It is important not to confuse the previous subsection about local searches which complete in polynomial time with the current topic. The perhaps unfortunately named PLS class is about problems in which a *single step* of local search takes polynomial time, and includes problems for which the number of such steps performed during a local search is *not* necessarily polynomial.

The main result about PLS problems is that simply finding *any* local optimum (as defined by the algorithm C) can take longer than polynomial time. The class of search problems for finding any local optimum of a PLS problem is also known as PLS. It is known that PLS contains P_S, the class of search problems computable in polynomial time, but it is thought that PLS is not equal to P_S [45].

This fact does not directly address the issue of using LS in conjunction with an EA. In an EA setting, we would like to know how quickly LS (or some other algorithm) can find the *particular* local optimum associated with a given starting point (under the local search algorithm defined by C, for instance). The PLS result does at least tell us that this cannot, in general, be done in polynomial time.

In fact, it is known that for some problems, there are instances and starting points for which local search takes exponentially many steps. Such instances arise, for example, in TSP under the 2-change neighborhood [58]. Hopfield neural networks are also subject to potentially exponential settling times ([65, Chapter 10] and [64]). Even worse for practitioners, it turns out there exist PLS problems and starting points s for which it is NP-hard to find the local optimum associated with s, given a function

C. In other words, it's not just that the local search procedure is brain-dead; no algorithm can perform its task in polynomial time (if $P \neq NP$). This is shown by a construction based on SATISFIABILITY in [45].

These results appear to counter our complexity argument for why EAs are suitable for choosing LS starting points. If LS itself can take more than polynomial time, then even if the EA works perfectly (e.g. quickly finds a starting point in an optimal basin) we may not succeed. However, this does not necessarily argue against the EA, as no other method for choosing starting points will alleviate this problem either. Furthermore, as noted in [45], these worst-case local searches seem only to arise for perverse or contrived instances and starting points: for example, in TSP using the 2-swap neighborhood, the only known examples of exponentially long LS sequences are for complex contrived instances, and no such instances are known for the λ-swap neighborhood with $\lambda > 2$. Empirically observed LS lengths from random starting points on more natural instances typically grow as low-order polynomials [57, 66]. Indeed, as we have shown in the previous subsection, under reasonable restrictions of common problems local search can be guaranteed to take only polynomial time.

III.B Operating Assumptions about EA+LS

This section will try to make explicit our assumptions about how EA+LS algorithms operate. We will consider the ways in which LS interacts with the global search, what roles it fills, and what it *should* do for us. For the most part these comments will apply to both the continuous and combinatorial cases, but exceptions will be noted.

1. LS searches within basins, the EA searches *over* basins. As previously noted, Hart has observed that this is not always true or even desirable in the continuous setting. It is a true statement, however, in the combinatorial case. The question then becomes how to ensure that the EA effectively chooses basins, while relying on LS to do refinement.

2. The EA should not generate multiple solutions in the same local basin. Any time LS is done from multiple points in the same basin there is the potential for wasted effort. For instance, in the case of complete LS and Lamarckian evolution we will end up with two population members which are exactly the same, and this is unlikely to help the global search. It will not be practical to entirely prevent such an event from ever occurring, but this principle can influence the design of the EA.

3. The minimum scale of the evolutionary operators should be the same as the size of local basins. This means that the genetic operators (especially mutation) should modify solutions at least enough to move them out of their current basins. This is related to the above point. Mutations which move solutions around within a basin have little effect, as LS can take any point in the basin back to the local optimum. In EAs without LS, mutation can serve the role of a refinement operator, but this is entirely inappropriate when LS is being used explicitly. In this case, mutation is probably best used to search one step higher in the scale hierarchy, exploring nearby local basins.

III.C Designing Complementary Operators

When LS is added to an EA, it interacts with the genetic operators and alters the dynamics of the search. The presence of LS also modifies the role that the other operators play, and allows these roles to be redefined to some extent. The role of LS, its interactions with the EA, and the implications of these issues are the topic of this subsection.

III.C.1 Role of LS

To understand how LS interacts with the EA, we must understand what it does. We also have some freedom in implementing LS to reflect what it is we want it to do. The first very important observation about LS is that it is often quite powerful in its own right. Frequently an effective method for global optimization is simply to do some sophisticated LS from a succession of randomly chosen starting points. This method has been used, for example, to do graph partitioning with the Kernighan-Lin algorithm [51], MAX-SAT with the GSAT algorithm [74, 53], and thin film metrology using a Levenberg-Marquardt least-squares fitter [56]. Even very simple LS algorithms can often be powerful. In graph bipartitioning on random graphs, for example, random local optima under the 2-swap neighborhood are virtually guaranteed to be substantially better than random non-optimized partitions.

First and foremost, LS is a *refinement* operator. It takes solutions and quickly refines them until no further (local) improvement is possible. Compared with the EA itself, LS is quite efficient at finding the local optimum of a basin. An EA will eventually get there, but it will take longer, as it proceeds through fortuitous mutations and occasionally selects less fit individuals for reproduction (thus moving in the wrong direction). EAs have long been known to be poor local searchers (cf. Section II.A.3).

Since an EA+LS algorithm will make use of the LS algorithm many times, it is important that the LS method be relatively quick. Generally this means using a small neighborhood. While this means the local optima found are likely to be inferior to those found under a large neighborhood, this is more than compensated for by the fact that many more local searches can be done (they may be used to have more generations or a larger population, for instance). Note that seemingly small changes in the neighborhood definition can result in dramatic changes in neighborhood size, and hence LS efficiency. For and n-bit string, there are n 1-bit neighbors but $O(n^2)$ 2-bit neighbors.

The speed of LS is also affected by the method used to decide which neighbor to move to. In general the first-improve method will be quicker than the steepest-descent method, which must always examine the entire neighborhood. To bring these issues into focus, consider some data gathered on a 1000-node geometric graph. An average 2-swap steepest descent LS examines over 62 million solutions before terminating, whereas an average first-improve 1-swap LS sees less than 20 thousand. This difference is so great that an entire EA+LS run using the quick method can complete before a single LS using the slower method. See Section IV.B.2 for a more complete description of this data.

How can we be sure the extra quality gained by a long-running LS isn't worth the extra effort? It may well be worth it if we are simply going to do LS from random starting points. But in our context, we are relying on the genetic operators to search over basins and LS to do refinement within basins. Large neighborhoods result in larger local basins, and hence make the LS more global. If the price of this is a vastly more expensive LS method, then we are better off relying on the EA for this global search. Even if the larger neighborhood is not more expensive, it is not clear that it offers a benefit in the face of an EA search which presumably will be able to examine nearby neighborhoods if the region looks promising. Note that this can be taken too far: the smallest non-empty neighborhood structure is for each solution to have exactly one neighbor. This would likely result in very small basins, and LS would be unlikely to offer substantial gains.

Another way to view this issue is to consider the tradeoff between LS effort and population size. A quicker LS allows a larger population, while using the same total search effort per generation. A larger population allows a better statistical sample of the search space and slows convergence of the population. These are both potentially substantial benefits to the global search. A larger population also better allows the EA to assume the role that a larger LS neighborhood would serve. Specifically, the EA is better able to explore nearby neighborhoods which would be coalesced under a larger neighborhood structure.

III.C.2 Mechanisms of Global-Local Interaction

There are a number of ways in which LS can alter the dynamics of an EA search, aside from its nominal role as a refinement operator. By returning the optimum of each basin it examines, LS allows a better characterization of the search domain. If the EA is searching over basins, then its ultimate goal is to find the basin whose local optimum is also the global optimum. Towards this end, it is the optima of the basins that would seem most important for informing the EA search. On a related note, consider the sampling that an EA does in each generation. In order to compare basins, it would desirable if population members in different basins were somehow similarly representative of those respective basins. Figure III.1 shows a situation where an unfortunate sample is misleading with regards to which basins are best. The local optima, of course, define the value of basins, and so cannot be misleading. In this way, LS allows a *more reliable sample* than random sampling without LS.

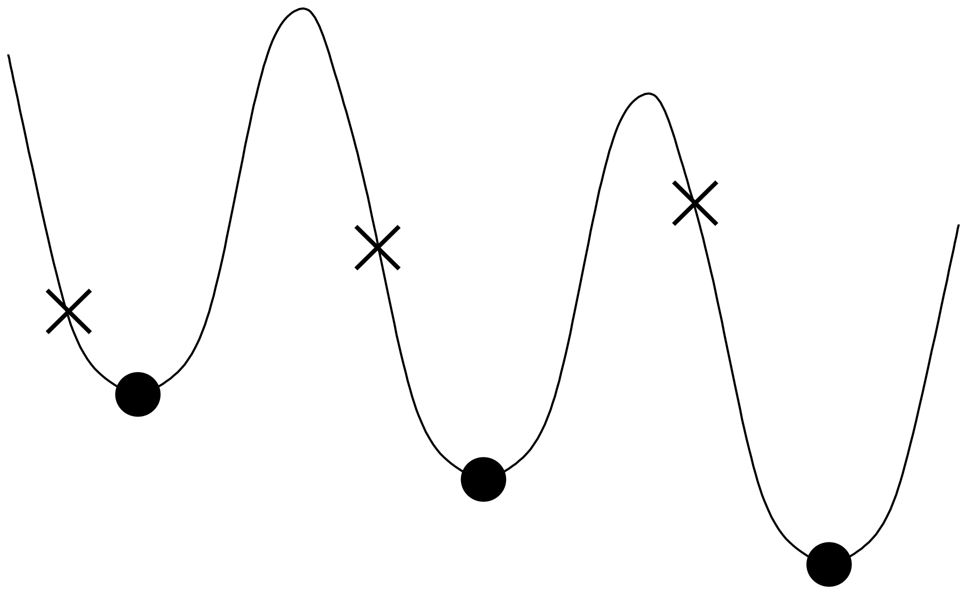

Figure III.1: **Misleading sample of local basins:** The solid dots mark the local optima of the basins. Sampling randomly allows the possibility of being misled about the relative goodness of the basins: the **X**'s in the diagram indicate sample points which are anticorrelated with the local optima.

Another way LS can affect the global search has to do with commonalities across local optima. For combinatorial problems local optima can be expected to have many features in common [9]. As an example consider a MAX-SAT formula that has (among others) several variables which occur exactly once. Under any standard neighborhood structure, these variables will always be set the same way by LS no matter what the initial assignment is. In a sense there is wasted effort in having LS discover these settings over and over throughout the course of an evolutionary run. But note that with Lamarckian evolution, these common settings will quickly come to dominate the population and then fixate. In this sense the dimensionality of the global search will be reduced to the non-trivial variables. We believe that this is generally a beneficial effect.

The power of LS can sometimes disrupt the genetic search by leading to an unwarranted loss of diversity. In the case of infrequent LS (only a fraction of the population gets LS in any given generation), the solutions which do get LS will likely end up with much better fitness than the others. The fitness can be so much better that these solutions come to dominate the population. They may or may not actually represent better basins, but they "drag" the population in their direction nonetheless.

Competing with this tendency is the notion of "delayed commitment" which can occur with Darwinian evolution. Consider a solution which is in a good basin but which is not the local optimum. Its fitness will remain the same (equal to the local optimum) even if it gets modified by genetic operators, as long as it remains within the basin. In effect, this allows it to explore the entire basin without suffering from reduced fitness. Why might this be helpful? This solution can bump into all the adjacent basins, in effect giving it an expanded neighborhood. It can survive while it explores a large area. This is sometimes referred to as the *Baldwin effect*.

III.C.3 Biasing LS Selection

When doing LS on only a fraction of each population, some method is needed to choose those solutions which will undergo LS. Usually they are chosen uniformly at random, but there are more sophisticated possibilities. We will examine three other methods of biasing LS selection: based on fitness, diversity, and "LS potential."

Fitness An obvious method for local search selection is to use the same procedure that reproductive selection uses, namely biasing towards the selection of 'fit' solutions.

Upon first inspection this seems to make good sense. A strawman argument in favor of this approach might go as follows: "This approach focuses LS on the best solutions, which are most likely to be in the best local basins. Also, the EA will be generating the next population from these good solutions anyway, so it doesn't make sense to use LS on the weak solutions which are less likely to influence the next generation."

This argument fails because it ignores the dynamics of the EA, and the *interaction* between LS and the global search. The argument might hold if LS were to be applied during a single generation only, but the cumulative effect of using LS on successive generations makes the analysis more complicated. Consider, the best solutions in the population are likely to be those which have already had LS, or whose parents benefited from LS.[1] They may be good simply by virtue of previous LS, and do not necessarily represent a better region of the search space. Biasing additional LS towards these solutions reinforces their dominance.

In this way, search focuses on the portion of the search space which happened to get LS early on. The result is a reduced chance of exploring novel regions of the search space. This can be expected to lead to quick convergence to a final solution, but at the expense of solution quality. Additionally, if the best solutions have indeed benefited from LS then they are more likely to be near local optima.[2] This implies that these solutions are the ones which will get the least benefit from LS, and will also be the least "efficient" to optimize. This is because LS generally needs to examine more neighbors to find an improving solution when it is near an optimum.

The above discussion can be summarized by a set of predictions about the performance of an EA+LS using this method of local search selection. In comparison to the random local search selection method, fitness-based local search selection should result in faster convergence but worse solutions, on average. Furthermore, since the course of the EA is more heavily influenced by the initial allocation of LS,

[1]For example, in bipartitioning the effect of LS is so great that random solutions are almost certainly substantially less fit than randomly selected local optima.

[2]These solutions are not necessarily exactly at local optima already. They may have been generated through mutation or recombination of locally optimal solutions, or they may have undergone partial LS.

we can predict a greater variance in solution quality from one run to the next.

An interesting alternative to consider is to select *less fit* solutions for LS. This will likely have the effect of allocating LS to solutions which have not benefited from it before, and hence spreading LS to different regions of the space. In contrast to the above situation it would allow a 'fairer' comparison between diverse solutions and result in greater exploration. However, it is hard to see how this could be superior to random LS selection, which also allocates LS to diverse regions of the space, and without 'rewarding' bad solutions.

Diversity In light of the above discussion of how solutions which get LS can come to dominate the population, it is desirable to explore ways to prevent this. In particular, we would like to ensure that solutions from different regions of the search space all get LS, otherwise the EA may bias succeeding generations towards a particular region simply because the solutions in other regions did not get LS. One way to approach this is to use an explicit *diversity-based* LS selection method: selected solutions will be far away from each other, and ideally span as much of the search space as the population itself. This helps ensure locally optimized solutions cover the search space, and tends to prevent premature convergence.

Stated another way, the proper scale for the global search is the current span of the population, and for this reason doing LS on two relatively nearby solutions may be inappropriate. During any given generation, the EA needs to be able to compare regions as diverse as its population, and is less concerned with distinctions between relatively local variants. Furthermore, selecting diverse solutions reduces the chance that two points from the same local basin will undergo LS. This would be something of a waste of effort, especially in the Lamarckian case, as we would have two copies of the same local optimum in the population.

There are several possible ways to select a diverse set from the population. This is very similar to the diversity enforcement [60, 59] and fitness sharing [28, 61, 77] schemes which are often used for reproductive selection in EA practice. This case is different, however, in that we are selecting a small subset of the population (without replacement). In contrast, selection for reproduction typically chooses a set as large as or larger than the population, *with* replacement.

One approach explored by Hart [38] is the use of a generalized F-statistic. In biology, the F-statistic is used to measure the degree of inbreeding in a population. Hart adapted this notion to the case of haploid genotypes and used it to identify diverse solutions which would become more likely to receive LS. More recent discussions with Hart [32] have resulted in various other possible approaches.

To describe these approaches formally, we introduce some notation. Let P be the current population. Let L be the set of solutions to be selected, on which LS will be performed. Let $|P| = n$, $|L| = k$, and d_{ij} be the distance (according to some metric on the search space) between solutions i and j. Consider the following three methods to select L.

1. One of our goals is to choose points which are as far away from each other as possible, so that no two selection points are near to each other. Formally, we want choose the set which maximizes the minimum distance between points in L:

$$L = \arg \max_{L \subset P} (\min_{i,j \in L} d_{ij})$$

2. An alternative priority is to ensure that every point in the population is near some solution which will get LS. For instance, if the population consists of distinct but tight clusters, we would ideally choose one solution from each cluster. This goal is not quite the same as the above, as it takes into account the distances between selected points and nonselected points.

   This goal can be formalized as follows: Let L be the set which minizes the maximum distance between any nonselected solution and its nearest selected solution.

$$L = \arg \min_{L \subset P} (\max_{i \notin L} \min_{j \in L} d_{ij})$$

3. The above two methods exactly specify L, but may not be efficient to compute. The straightforward algorithms require $O(n^k k^2)$ and $O(n^{k+1}k)$ time. A linear time heuristic method is to iteratively select k points from P, making sure each new point is at least a distance d from any previously selected point. The parameter d would need to be set carefully at each generation, based on the current population. Setting it too high would cause the above procedure to fail, whereas setting it too low would allow choices of L which do not span the population.

Whichever of these methods is used, a basic prediction can be made about the behavior of an EA+LS using a diversity-based LS selection scheme (in comparison to random LS selection). Because diverse solutions are explored, it should take longer for the population to converge, and the final solution can be expected to be of better quality, on average.

LS Potential Ultimately, the motivation for biasing LS selection is to choose solutions for which LS will be most "useful.". Roughly, the usefulness of LS on a particular solution is based on how much information we gain about that solution and what this tells us about the global search. The latter is usually the more important consideration in an EA context, especially during the early stages of a run. As an example, performing a full LS on one solution is not as useful in characterizing the global landscape as performing half-completed LS on two distinct solutions. The former tells us about a single point only, and does not allow a meaningful comparison of that locally optimal point with other nonoptimized points. The latter at least allows a fairer comparison of two points from different regions.

Along with usefulness, the efficiency of LS must also be considered; solutions that are near their local optimum likely require more computational effort per unit gain in fitness than solutions which still have many improving neighbors. These issues motivate the use of a "LS-potential" LS selection technique, in which solutions are chosen according to their expected gain in fitness per unit of computational effort.

This makes most sense in the context of partial LS, where only a given (small) amount of computational effort is expended for any one application of LS. The continuous analog of this technique would be to choose the solutions with largest gradient. In the discrete case, we mean the expected gain in fitness over a small fixed amount of LS (for example, 20 function evaluations), what we term *LS potential.*

The idea is that if it can be accurately determined which solutions can be most efficiently improved, then these solutions will make the most effective use of LS. There are several complementary reasons why this should be so. The most straightforward is that allocates LS to the solutions on which it can be of most direct benefit. In terms of simply improving average population fitness, these are the optimal solutions to choose. Conversely, the least easily improved solutions are likely to be those which are at or near local optima.[3] Such solutions have few improving neighbors and so LS must examine many neighbors at each step.

Indeed, applying LS to these nearly optimal solutions is not likely to be very helpful. The small gain in fitness which may be achieved is probably dominated by the difference in fitness values of the various solutions in other local basins. As long as there are large differences between populations members which have not all been optimized, it is inappropriate to expend effort on minimal refinement. The proposed method will apply LS to less optimized solutions until such time as *all* members of the population are nearly optimal. More generally, it allows all solutions to progress to roughly the same stage of LS in their respective basins, thus facilitating comparisons between them. As the population becomes more stable and the solutions get closer to the optima, this scheme automatically allows LS to progress to the next level of refinement, in a sense adaptively adjusting the fitness scale of refinement.

The above discussion assumes some way of determining LS potential. How can we estimate this in practice? A simple method would be to invert the fitnesses, assuming that the most fit solutions are most difficult to improve, and the least fit are easiest to improve. This assumption may be roughly true in an average sense,

[3]Obviously we should not apply LS to a known local optimum.

but it is very crude. In reality, there are certain to be local optima which have poor fitness, and good solutions which nevertheless have can be much improved. Note that this method is the same as the inverse-fitness LS selection method discussed above.

A more sophisticated method involves "sniffing" the potential of each solution as it is produced (though mutation or recombination). Each solution undergoes a small amount of LS (e.g. 20 function evaluations) and its fitness improvement is recorded. This gain per unit of effort is taken to be its potential. In other words, we use a measure of past LS effectiveness as an estimate for future effectiveness. The effect of any future LS is recorded, and the LS potential is always taken to be the LS effectiveness over the most recent k function evaluations for that solution. In this way every population member has a LS potential associated with it which can be used to allocate future LS.

This method has a couple of drawbacks. The first is that is may be necessary to do a fair amount of LS to get a reliable estimate of future potential. The second is that it may deal poorly with "saddle points," solutions which have few improving immediate neighbors, but which have much room for improvement a few steps away. The initial estimate may label such solutions as having very low potential, which would be inaccurate.

III.C.4 Reconsidering Standard EA Operators

In light of the effect of LS, the standard EA may need to be reconsidered. Most obviously, small mutations don't make sense if each population member will be locally optimized each generation. Any changes caused by mutation away from a local optimum will be undone by LS. In this case mutation can have no effect within a basin. This suggests using larger mutations, at least large enough to move from one basin to another. Such an operator has exploratory value, and in fact is similar to (small) mutation's original role, only on a larger scale.

Chapter IV

Experiments

This chapter describes a collection of experiments and discusses the results. The test problem for nearly all experiments is binary graph bipartitioning (or bisection), which is described in Section IV.A along with the instance distributions used and other details related to our experiments. In the remaining sections we compare the EA+LS algorithm to Monte Carlo local search, simulated annealing, and the EA without local search. In Section IV.D we describe a variant of the steady-state EA which allows fine control over various EA parameters that are relevant to local search. We then present a comprehensive examination of these parameters and how they impact the effectiveness of the EA+LS.

Most experiments in this chapter are performed on a small (eight instances) collection of graphs of various types. We will generally be concerned with trends across the graphs (e.g. method A works best on all graphs, or works best only on geometric graphs, etc.). After describing the instance classes we use (Section IV.A.1) and investigating some general properties of the graph bisection search space (Section IV.A.2), we describe the details of our experiments and data presentation in Section IV.A.3. This sets the stage for the rest of the chapter.

IV.A Binary Graph Bipartitioning

The main test bed for this dissertation is the problem of binary graph bipartitioning. This is a well-studied problem in combinatorial optimization which is difficult enough to be of interest, yet allows a simple representation for evolutionary algorithms. It is an important problem in the field of load balancing for parallel computer systems, as well as for the placement of circuit components. There is also a well-known and comprehensive study by Johnson et al. [44] regarding the effectiveness of simulated annealing on various instances of this problem. In Section IV.E.1 we compare the performance of the EA+LS with simulated annealing as described by Johnson et al.

The general graph k-partitioning problem is defined for an undirected graph G with weights edges E and nodes V. The goal is to find a *partition* of the nodes into equal-size subsets so as to minimize the sum of the weights of all edges which connect nodes from different subsets. More formally, define a k-partition of V to be a set of k subsets, $V_i \subset V$, for $i = 1 \ldots k$, such that $\bigcup_{i=1\ldots k} V_i = V$, $V_i \bigcap V_j = \emptyset$ for $i \neq j$, and $|V_i| = |V|/k$ for $i = 1 \ldots k$. Then the problem is to find a k-partition of V which minimizes

$$\sum_{v \in V_i, w \in V_j, i \neq j} w(v, w).$$

We will be looking a subclass of this, namely binary graph bipartitioning. In this case, partitions contain only two subsets ($k = 2$), and all edge weights are either zero or one. We will sometimes say a graph has or doesn't have a particular edge to mean the edge has a weight of one or zero, respectively.

Both general graph partitioning and the binary bipartitioning version are NP-complete, but there are positive results concerning how good an approximation is possible with polynomial-time algorithms. Saran and Vazirani show that the general problem is approximable to within a factor of $|V|/2$ [72]. A more encouraging result due to Arora, Karger and Karpinski [4] is that for binary graphs in which every vertex has degree $\Theta(|V|)$, bipartitioning has a polynomial-time approximation scheme. An interesting question is what happens to the complexity of the problem when a penalty term is added for unbalanced partitions. Note that if the weight on the penalty is low enough (e.g. zero) then the problem is equivalent to MIN-CUT, which is in **P**.

Methods for solving graph bisection can be classified according to how quickly they run and the quality of the solutions produced. The quickest methods which produce high-quality solutions simply apply local search from randomly chosen starting points. Section IV.B explores this technique in some detail. Some fairly recent techniques which quickly produce very good solutions are hierarchical clustering [40] and spectral methods [75, 67, 10, 39]. When speed is not crucial and it is important to find the best partition possible, simulated annealing is generally used. Like simulated annealing, the EA+LS is a long-running method which aims to explores to entire domain. Hence, we consider simulated annealing to be the most appropriate method to compare against. This comparison is made in Section IV.E.1. Go-With-the-Winners is an even slower technique which may be able to find even better solutions. We compare this to the EA+LS in Section IV.E.2.

There is a natural representation for graph bipartitioning in the context of an evolutionary algorithm. The genotype is simply an array of $|V|$ bits, each bit indicating which subset that node is a member of. There are a couple of issues arising from this simple scheme. The first is that there are two equivalent representations of any partition; flipping all bits gives the same partition but results in a bitstring which is maximally far away in terms of Hamming distance. Where we discuss such distances, we will often "normalize" partitions so that they are closer together. The second issue regards the fact that the representation of a valid partition must have an equal number of ones and zeros. Generic genetic operators such as single-point crossover or bit-flipping mutation do not preserve this property, and so produce invalid partitions. We will have to use more specialized operators to avoid this. See Section IV.C.2 for details.

When doing local search, we will usually be searching over the *unbalanced search space*, which includes invalid partitions (unequal subset sizes) as well as valid ones. In order to bias search towards the balanced partitions, a penalty is added to the fitness which increases with the degree of imbalanced. Following Johnson et al., the penalty we use is $0.05(|V_1| - |V_2|)^2$. The leading constant 0.05 was found to allow for good good solutions to be quickly found in the context of simulated annealing [44].

IV.A.1 Graph Instance Distributions

We examine partitioning on three classes of binary graphs, *random*, *random geometric*, and *planted bisection* graphs. Several instances of the first two were studied in [44]. All of those instances are among those we examine with local search (cf. Section IV.B), and several of them are used for the comprehensive examination of EA+LS in Section IV.D. Planted bisection graphs are more easily understood, and have been analyzed by Carson and Impagliazzo [13].

Random A random graph of size n is generated by considering all pairs of nodes, and including an edge for each pair with some fixed probability p. The expected degree for each node is then $(n-1)p$. Johnson et al. use graphs of size 124, 250, 500, and 1000, with expected degree 2.5, 5.0 10.0, and 20.0 for each size.

Geometric To generate a random geometric graph of size n, n points are generated uniformly at random over the unit square. Each point corresponds to a node in the graph. Any two nodes which are within Euclidean distance d of each other are connected by an edge. The expected average degree is related to the parameter d. Note that a circle of radius d inside the square is expected to enclose $n\pi d^2$ points, so this is a rough approximation of the average degree for relatively small values of d. Points nearer to the edge of the square will obviously have smaller expected degree. Johnson et al. use graphs of size 500 and 1000, with expected average degree 5.0, 10.0, 20.0, and 40.0 for each size.

Planted Bisection A planted bisection graph is one in which a desired bisection is purposefully embedded in an otherwise random graph. Specifically, each pair of nodes is connected by an edge with probability q if the nodes are on opposite sides of the partition, and probability p if on the same side. By following this procedure with $p > q$, we can generate graphs which are biased towards the desired partition having minimal cost. If p is sufficiently large in comparison to q, then with high probability the desired partition is the actual global minimum [13]. Such graphs have a very simple global structure, and the global optimum can often be found by performing local search from random initial partitions. Carson and Impagliazzo have described ranges of p and q for which this is least likely [14]. We examine two graphs, of size 250 and 500, with $p = 32/n$ and $q = 18/n$ for each. Hence, these graphs have average expected degree 25, (16 edges on the same side of the partition, 9 going across the partition).

In preliminary experiments using local search (Section IV.B), we examine all 24 graphs from the Johnson study, the two planted bisection graphs mentioned, plus one additional random geometric graph with size 250 and expected degree 20.0. For the more comprehensive EA+LS experiments (Section IV.D), we examine the small (500 nodes or fewer) random and geometric graphs with degree 20 as well as the two planted bisection graphs.

From this point on, the following notation will be used to refer to the various graphs: a single letter denoting the type of the graph (r for random, g for geometric, or p for planted bisection), followed by the number of nodes in the graphs, followed by the expected average degree. For example, the geometric graph with 500 nodes and degree 20.0 is referred to as "g0500.20."[1] The planted bisection graphs are designated "p0250.9+16" and "p0500.9+16," to emphasize the distinction between edges going across the partition and those on the same side.

[1]Note that this notation is somewhat different from that used by Johnson et al. In particular, they use g and u for the random graphs and geometric graphs, respectively. Furthermore, for the random graphs, their label specifies the value of p used instead of the expected degree. Hence, their g0500.04 is our r0500.20.

IV.A.2 Free Nodes and Node Affinities

An important feature of all the graphs we examined is that their search spaces possess plateaus of same-fitness partitions. When considering the unbalanced search space with fitness penalty, these plateaus translate to regions with many balanced local minima of equal fitness separated by small barriers. These regions are problematic for local search, as it quickly gets stuck in one of these tiny "artifactual" local basins, as opposed to exploring the real structure of the search space. Section IV.D.3 illustrates how this leads to difficulty in defining local basins and determining their characteristic sizes.

Free Nodes Consider the case of *free nodes*, or nodes of degree zero. Several of the low-degree random graphs used by Johnson et al. have many free nodes. In any partition in which the set of free nodes is split among the subsets of the partition, the free nodes can be moved about or interchanged without affecting the cost of the partition. Furthermore, the resulting partitions are not different in any way that is relevant to the global search. Figure IV.1 displays part of a graph with free nodes, and a corresponding fitness landscape. Free nodes a and b can be placed on either side of the partition without affecting the cost, other than perhaps incurring the imbalance penalty.

If there are k free nodes to split evenly by the partition, there will be equivalence classes of partitions, each with $\binom{k}{2}$ partitions having the same cost. These equivalent partitions are adjacent to each other in balanced search space, and are two moves apart in unbalanced space. In unbalanced space, the immediate neighbors of any balanced partition include equivalent partitions with the same cost, except that they are penalized by the imbalance penalty. Hence, there may be a "plateau" with many equivalent size one local basins, artifacts of the unbalanced space and penalty. The bottom half of Figure IV.1 shows such a situation. Unfortunately, local search will settle into one of these and stop instead of exploring the structure of the actual (balanced) partition space.

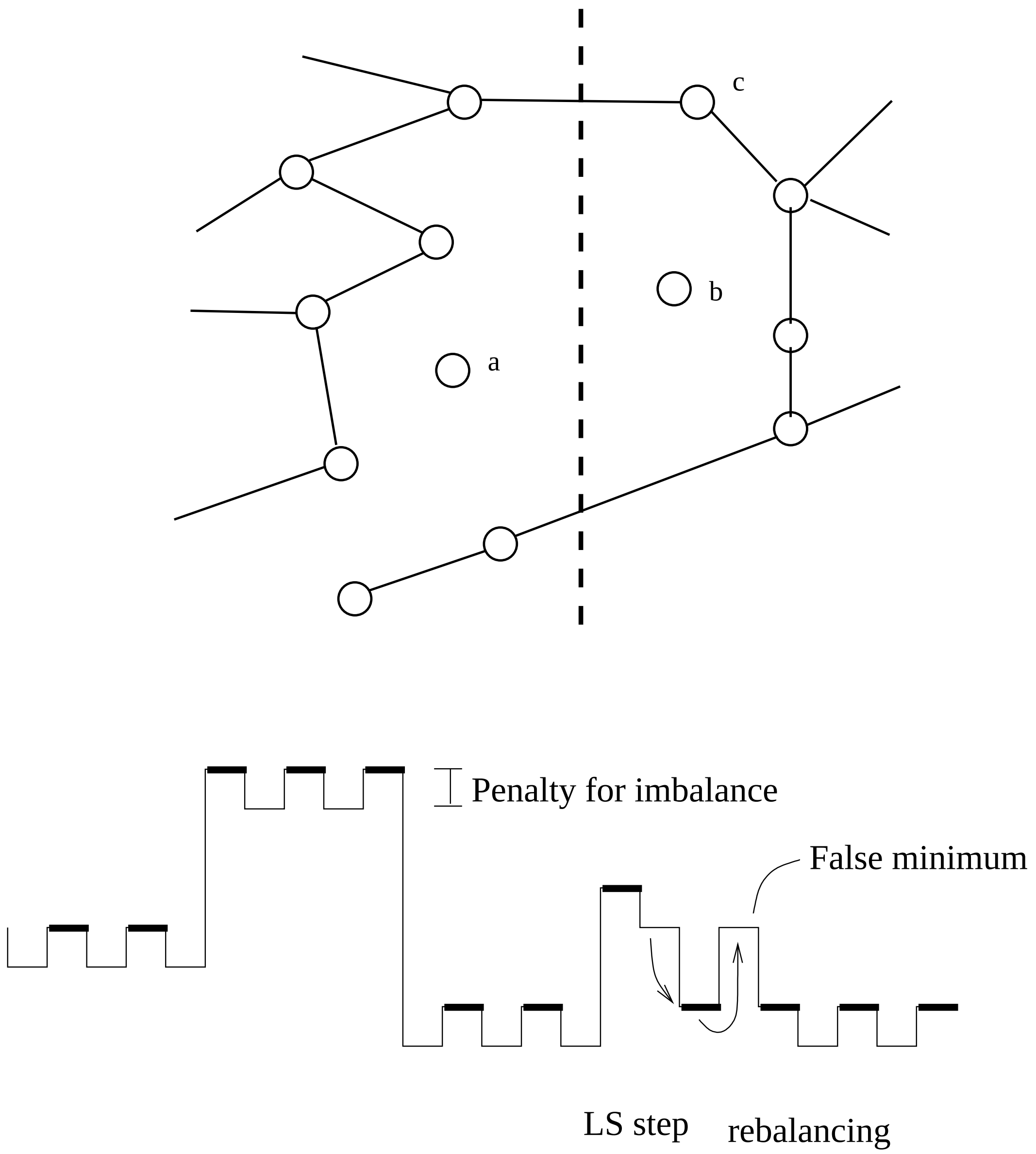

Figure IV.1: **Part of a graph and fitness landscape:** Nodes *a* and *b* are free nodes, while node *c* has affinity zero. The bottom half of the figure displays a fitness landscape, with unbalanced partitions represented with bold lines.

This effect also complicates the analysis of graphs and certain algorithms such as simulated annealing. In Section IV.E.1 we examine the "basin-finding behavior" of simulated annealing, how frequently it switches basins and discovers new ones as temperature decreases. The proliferation of equivalent basins makes it more difficult to interpret the results there. Alternatively, in Section IV.D.3 we attempt to measure the average basin size of various graphs in order to better design genetic operators. This endeavor is also complicated by the same effect.

Node Affinity Besides nodes of degree zero, there may be other "pseudo-free" nodes in a graph which can move between sets without affecting the partition cost, depending on the current partition. These nodes lead to a similar effect as that described above. Significantly, however, moving these nodes around may affect the possibilities of future improvement, and so the resulting partitions are not truly equivalent in terms of the global search. We will define a *node affinity*, where nodes with affinity zero play a similar role to the degree zero nodes above. These nodes arise both for the low-degree graphs and also the higher-degree graphs, which have no nodes of degree zero.

In the context of a given partition of a graph, a node's *affinity* is the number of edges it has to nodes on the same side of the partition, minus the number of edges to nodes on the the opposite side:

$$\mathit{affin}(v) = |v, w \in E, w \in V_1| - |v, w \in E, w \in V_2|,$$

where $v \in V_1$. Alternatively, it is the change in cost associated with moving the node to the other side of the partition. Intuitively, this describes the affinity a node has for the side of the partition it is currently in. Degree zero nodes always have affinity zero, whereas the affinity of all other nodes will depend on the current partition. Local search will only move nodes with $\mathit{affin} \leq 0$. In this sense, affinity also describes how much about the current partition has to change before a given node can be moved. The concept also plays a role in simulated annealing, where the probability of moving a node is inversely correlated with its affinity. In Section IV.E.1 we present results confirming these intuitions.

Nodes with *affin* $= 0$ are interchangeable just as nodes of degree zero are, subject to the possible change in affinities once nodes start moving (for example, two nodes with zero affinity can each be moved independently without affecting the cost, but if they are connected, then moving one will affect the affinity of the other). Therefore, they lead to similar regions of degenerate local basins containing roughly equivalent partitions. In this case, the different partitions may actually be different with regards to the global search. Nevertheless, the same problem of local search getting stuck quickly results. A couple of examples of common situations where node with zeros affinity arise are shown in figures IV.2 and IV.3.

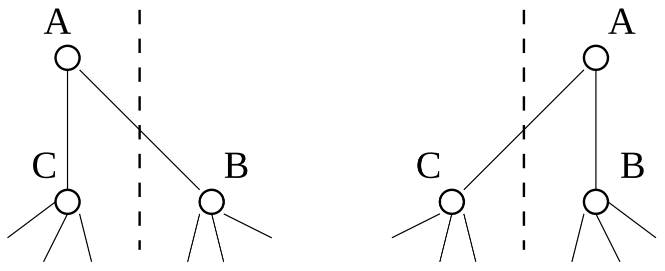

Figure IV.2: **Node with affinity zero:** Node (A) has degree two, and is connected to neighbors (B and C) in opposite sets. Node A switches sets without changing the cost. The partition cuts vertically, so that nodes to the left and right of the dashed line are in opposite sides of the partition. B and C may be connected to other nodes not shown, but A is not. A therefore has affinity zero.

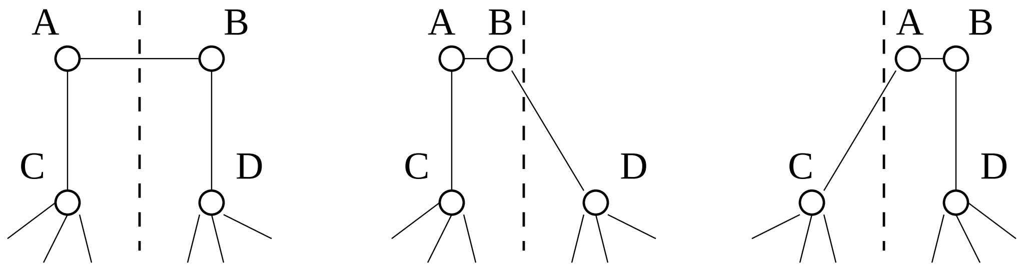

Figure IV.3: **Pair of nodes with affinity zero:** Pair of nodes (A and B) connected to each other, and to nodes (C and D) on opposite sides of the partition. Nodes A and B switch sets without changing the cost. A and B both have affinity zero in the leftmost diagram. Note that longer chains of degree two nodes would allow even more partitions with equal cost.

IV.A.3 Experimental Details and Data Presentation

Most experiments in this chapter are performed on a small (eight instances) collection of graphs of various types. We will generally be concerned with trends across the graphs (e.g. method A works best on all graphs, or works best only on geometric graphs, etc.). Hence, data for all eight graphs will be plotted side-by-side in a single figure (see Figure IV.4 for an example). The plots are arranged so that the three plots on the upper left side are for the random graphs, with graph size increasing down the page. The three plots on the upper right side are for the geometric graphs, with graph size and/or degree increasing down the page. Finally, the bottom two plots are for the planted bisection graphs. These classes of graph instances, as well as the particular instances used, are discussed in Section IV.A.1.

The data displayed in the eight-plot figures is usually the performance of EA+LS as a function of time. Unless otherwise stated, the data shown will be the fitness of the best solution in the population, averaged over ten runs. For a given algorithm and graph instance the ten runs will differ only in the initial random seed used, but the same group of ten seeds is used for all groups of runs. When present, error bars will always represent the standard error over the ten runs (note that error bars are almost always used if not otherwise stated, though they are often too small to be seen). Wherever we report significance results for comparisons on a single graph, we use a one-way analysis of variance (two-tailed Student's t-test) with confidence threshold 0.05.

Each figure also displays the best known solution and the average solution found by simulated annealing, when these are available. Specifically, simulated annealing data is available for the three random graphs (r0124.20, r0250.20, and r0500.20) and the two larger geometric graphs (g0500.20 and g0500.40). The average SA solution is displayed with a dashed horizontal line for these graphs. For the geometric graphs, Johnson et al. [44] have developed specialized bisection techniques which find substantially better solutions than simulated annealing. The fitness of these solutions is displayed with a solid horizontal line for g0500.20 and g0500.40.

Finally, we can calculate the expected fitness of the best solution for the planted bisection graphs. This value is displayed by a solid horizontal line for two p0250.9+16 and p0500.9+16.

When considering performance as a function of time, our metric of time is the number of *function evaluations* used. Here, we consider every solution examined to be a function evaluation. Note that for many problems (including graph partitioning) there are efficient ways to incrementally update solutions when small changes are made (as in local search or simulated annealing, for example), so that the computational effort required to evaluate solutions may vary greatly depending on how the solution is created. Nevertheless, we stick with function evaluations as our metric, as we hope for our conclusions to speak generally about global/local search issues rather than about graph partitioning in particular. The number of solutions considered is a concrete metric which can be applied to any problem in a straightforward manner. Furthermore, even for comparisons within graph partitioning, we can easily compare general methods without regard for the implementation details of the algorithms employed.

IV.B Monte Carlo Local Search

In order to determine the properties of the graph instances, as well as the various local search methods, a series of "Monte Carlo local search" experiments was performed. For several graph instances, six local search methods are applied to 1,000 randomly generated partitions. Measurements gathered about each local search include the number of partitions considered and the decrease in partition cost.

IV.B.1 Local Search Methods

Three neighborhood structures are examined, and both steepest descent and first-improve search techniques (described in Section II.A.2) are examined for each, giving a total of six LS algorithms. The three neighborhood structures used are balanced, unbalanced, and half-balanced.

Balanced Under the *balanced* neighborhood structure, the neighbors of a partition are simply all partitions which can be reached by swapping any two nodes from opposite sides of the partition. In this case, all neighbors of a balanced partition are also balanced, and the search is conducted over balanced partitions only. Since each side of the partition has $n/2$ nodes, the neighborhood size is $(n/2)^2$, where n is the size of the graph.

Unbalanced The *unbalanced* neighborhood structure allows as neighbors all partitions (balanced or unbalanced) which can be obtained by moving a single node to the opposite side of the partition. A penalty term is added to the cost of unbalanced partitions (with penalty increasing quadratically with respect to degree of imbalance; see Section IV.A for details) to focus the search on "nearly balanced" partitions. In practice, however, the search can proceed for many steps without examining any balanced solutions. Note that this neighborhood structure has much smaller neighborhoods than the balanced structure (size n vs. $(n/2)^2$), and hence can be expected to allow for quicker local search.

A few important subtleties arise from allowing unbalanced partitions. First, there is no guarantee that the final local minimum found under this structure will be a legal (i.e. balanced) partition. In order to get a legal solution, then, the final unbalanced partition is brought back into balance one node at a time. This is done greedily, so that at each step, the node from the larger side of the partition which results in the best partition cost is moved. It sometimes happens that the balanced solution obtained in this way is inferior to a balanced partition which was seen earlier during the local search (this can happen because the search through unbalanced solutions reached a portion of the search space with inferior balanced solutions, or simply due to the imperfect nature of the greedy rebalancing). For this reason the best balanced partition seen during the local search is retained, and returned if it is better than the final (rebalanced) partition.

In this sense the final solution returned by local search may not actually be a local minimum. The fitness landscape of Figure IV.1 is illustrative. If the local minimum found has to be rebalanced, the resulting balanced partition will likely have worse cost than the unbalanced local minimum. Moreover, it will likely not be a local minimum in the unbalanced neighborhood structure. It may even be the case that performing an additional local search from this final partition would result in a better balanced partition. In the figure this is labeled as a "false minimum." However, since LS is nondeterministic, it would not be possible to check for false minima without performing all possible local searches, of which there could be combinatorially many. For this reason, no attempt is made to reoptimize or verify the final result of a local search.

Half-balanced The third neighborhood structure examined, the *half-balanced* structure, is a compromise between the balanced and unbalanced structures. For a balanced partition the neighbors are the same as for the unbalanced structure, namely all partitions which can be obtained by moving a single node to the opposite side of the partition. For partitions which are a single node out of balance, the neighborhood consists only of those *balanced* partitions which can be obtained by moving a single node. In other words, as the local search proceeds, the current solution will be alternate between being balanced and being one node out of balance.

The half-balanced structure has the same quickness advantage as the unbalanced structure, due to its small neighborhood size: balanced partitions have the same neighborhood under both structures (size n), but unbalanced partitions have an even smaller neighborhood under the half-balanced structure (size $n/2$). It also has the advantage of keeping the search tightly focused on balanced partitions, avoiding some of the problems mentioned for the unbalanced structure. It should be noted, however, that even this structure allows for the possibility of the final partition not being an actual local minimum.

The six local search methods are described by the three neighborhood structures balanced (**B**), unbalanced (**U**), and half-balanced (**H**), under either steepest descent (**S**) or first-improve (**F**). Henceforth, the six methods will be labeled **BS**, **BF**, **US**, **UF**, **HS**, and **HF**.

IV.B.2 Results

The experiments were performed on 27 separate graphs. These include the 24 graphs used in [44] plus three more generated for purposes of this study. For each graph instance, the six local search methods described were applied to 1,000 randomly chosen initial partitions.

Properties of graphs

Tables IV.1-IV.18 show the partition cost (average and best), average number of evaluations, and the number of distinct local minima found (out of 1,000 local searches) for each graph and local search method. These tables allow an easy comparison of the characteristics of the graphs as their size and degree vary.

The first thing to notice is that the number of evaluations used (equivalently, the number of partitions considered) per local search increases with both the size and average degree of the graph. With one exception, this is true for random, geometric, and planted bisection graphs. The number of evaluations used can be taken as a measure of the difficulty of the graph for local search. That more evaluations are required as the number of nodes increases is to be expected, as the neighborhood size scales (linearly or quadratically depending on the LS method being used) with number of nodes.

What is more surprising is that increasing the expected degree also increases the number of evaluations. For a given graph size, the neighborhood size is the same no matter what the average degree, so this effect must be due to an increasing number of improving moves made during local search. This implies that the local basins are larger as the average degree increases, and therefore that there are fewer of them. The one exception to this trend occurs with the half-balanced first-improve LS on

Table IV.1: **Average number of evaluations for** *HF*: The average, over 1,000 initial starting points, of the number of evaluations used (i.e. partitions considered) for the half-balanced first-improve LS method. The three groups of figures are for random graphs, geometric graphs, and planted bisection graphs. In each group, rows represent graphs with the same number of nodes, and columns represent graphs with the same expected degree.

	$\|V\|$	average degree			
		2.5	5.0	10.0	20.0
random	124	577	721	806	782
	250	1364	1748	2083	2231
	500	3199	3962	4935	5998
	1000	7418	9500	12099	15066
		5	10	20	40
uniform	250			1542	
geometric	500	3885	3917	3509	3309
	1000	9335	10436	9430	9157
			9+16		
planted	250		2259		
bisection	500		6234		

Table IV.2: **Number of local minima for** *HF*: The number of distinct local minima resulting from half-balanced first-improve LS applied to 1,000 random initial partitions. For the case in which fewer than 1,000 distinct local minima were found, the number of occurrences of the most common local minima is given in parentheses.

	$\|V\|$	average degree			
		2.5	5.0	10.0	20.0
random	124	1000	1000	999(2)	997(2)
	250	1000	1000	1000	1000
	500	1000	1000	1000	1000
	1000	1000	1000	1000	1000
		5	10	20	40
uniform	250			896(27)	
geometric	500	1000	1000	1000	958(13)
	1000	1000	1000	1000	999(2)
			9+16		
planted	250		1000		
bisection	500		1000		

Table IV.3: **Average/best partition cost for** *HF*: The average and best partition cost, over 1,000 initial starting points, resulting from half-balanced first-improve LS.

	$\|V\|$	average degree			
		2.5	5.0	10.0	20.0
random	124	24.4/ 15	77.8/ 64	196.0/ 180	479.1/ 452
	250	58.8/ 42	144.4/124	397.0/ 369	875.9/ 835
	500	105.1/ 86	291.5/260	705.2/ 658	1844.6/1778
	1000	210.6/182	590.2/543	1536.2/1469	3602.6/3500
		5	10	20	40
uniform	250			304.5/140	
geometric	500	69.2/34	180.4/ 70	466.2/192	1018.9/474
	1000	149.9/96	366.0/203	910.1/455	1860.2/737
			9+16		
planted	250		1183.8/1141		
bisection	500		2332.1/2260		

Table IV.4: **Average number of evaluations for** *HS*

	$\|V\|$	average degree			
		2.5	5.0	10.0	20.0
random	124	3284	3759	4140	4273
	250	11783	14578	16384	17612
	500	46287	54443	60840	69669
	1000	185793	212728	242092	272868
		5	10	20	40
uniform	250			21353	
geometric	500	67931	78774	85276	88285
	1000	261602	311360	342417	359776
			9+16		
planted	250		17996		
bisection	500		72015		

Table IV.5: **Number of local minima for *HS***

	$\|V\|$	average degree			
		2.5	5.0	10.0	20.0
random	124	1000	1000	1000	997(2)
	250	1000	1000	1000	1000
	500	1000	1000	1000	1000
	1000	1000	1000	1000	1000
		5	10	20	40
uniform	250			658(60)	
geometric	500	1000	1000	987(3)	760(34)
	1000	1000	1000	1000	954(10)
			9+16		
planted	250		1000		
bisection	500		1000		

Table IV.6: **Average/best partition cost for *HS***

	$\|V\|$	average degree			
		2.5	5.0	10.0	20.0
random	124	24.2/ 15	79.2/ 66	196.6/ 179	478.6/ 449
	250	59.3/ 45	146.9/126	400.9/ 370	878.0/ 836
	500	106.2/ 86	297.9/258	717.4/ 677	1856.0/1793
	1000	214.4/188	610.0/566	1567.2/1506	3636.4/3536
		5	10	20	40
uniform	250			237.7/140	
geometric	500	68.7/ 25	154.6/ 49	341.3/178	659.4/412
	1000	158.3/111	324.7/180	661.3/293	1226.9/737
			9+16		
planted	250		1186.4/1148		
bisection	500		2343.3/2278		

Table IV.7: **Average number of evaluations for** *UF*

	$\lvert V\rvert$	average degree			
		2.5	5.0	10.0	20.0
random	124	609	773	905	1160
	250	1426	1828	2247	2665
	500	3269	4066	5073	6591
	1000	7518	9454	12294	15929
		5	10	20	40
uniform	250			2619	
geometric	500	4164	4964	6010	7103
	1000	9503	11804	14470	19417
			9+16		
planted	250		2788		
bisection	500		6956		

Table IV.8: **Number of local minima for** *UF*

	$\lvert V\rvert$	average degree			
		2.5	5.0	10.0	20.0
random	124	1000	1000	999(2)	964(19)
	250	1000	1000	1000	1000
	500	1000	1000	1000	1000
	1000	1000	1000	1000	1000
		5	10	20	40
uniform	250			498(52)	
geometric	500	1000	1000	996(2)	409(106)
	1000	1000	1000	1000	941(5)
			9+16		
planted	250		1000		
bisection	500		1000		

Table IV.9: **Average/best partition cost for** *UF*

	$\|V\|$	average degree			
		2.5	5.0	10.0	20.0
random	124	24.4/ 15	77.6/ 64	195.1/ 179	474.2/ 449
	250	58.5/ 42	144.3/117	395.8/ 364	872.1/ 836
	500	105.2/ 85	290.8/259	705.9/ 666	1843.5/1782
	1000	210.4/177	591.0/540	1536.6/1469	3599.8/3510
		5	10	20	40
uniform	250			249.7/140	
geometric	500	68.0/35	166.1/ 60	386.5/178	758.6/417
	1000	148.3/89	355.2/203	821.1/400	1479.1/737
			9+16		
planted	250		1179.3/1128		
bisection	500		2330.8/2258		

Table IV.10: **Average number of evaluations for** *US*

	$\|V\|$	average degree			
		2.5	5.0	10.0	20.0
random	124	4331	5009	5464	6015
	250	15640	19307	21768	23501
	500	61481	72486	80809	93174
	1000	247324	283284	322700	365077
		5	10	20	40
uniform	250			29970	
geometric	500	90358	105168	117637	125487
	1000	348438	415185	462597	497522
			9+16		
planted	250		24029		
bisection	500		96218		

Table IV.11: **Number of local minima for** *US*

	$\lvert V\rvert$	average degree			
		2.5	5.0	10.0	20.0
random	124	1000	1000	999(2)	962(18)
	250	1000	1000	1000	1000
	500	1000	1000	1000	1000
	1000	1000	1000	1000	1000
		5	10	20	40
uniform	250			309(112)	
geometric	500	1000	1000	922(9)	182(317)
	1000	1000	1000	1000	739(29)
			9+16		
planted	250		1000		
bisection	500		1000		

Table IV.12: **Average/best partition cost for** *US*

	$\lvert V\rvert$	average degree			
		2.5	5.0	10.0	20.0
random	124	24.1/ 15	78.7/ 65	196.4/ 180	475.0/ 449
	250	59.1/ 45	146.7/125	400.1/ 374	876.9/ 833
	500	106.1/ 85	297.5/258	717.2/ 672	1854.5/1794
	1000	214.4/187	609.5/567	1566.4/1494	3634.2/3545
		5	10	20	40
uniform	250			215.0/140	
geometric	500	68.4/ 25	152.5/ 49	318.4/178	584.1/412
	1000	157.9/102	323.0/179	641.1/289	1151.6/737
			9+16		
planted	250		1184.6/1126		
bisection	500		2341.7/2275		

Table IV.13: **Average number of evaluations for** *BF*

	$\lvert V\rvert$	average degree			
		2.5	5.0	10.0	20.0
random	124	5179	6578	7740	9516
	250	18495	23861	31064	35939
	500	69634	78388	95227	138000
	1000	268784	289657	342934	461823
		5	10	20	40
uniform	250			40918	
geometric	500	98626	125650	153553	161510
	1000	330966	448410	578481	683845
			9+16		
planted	250		38598		
bisection	500		144486		

Table IV.14: **Number of local minima for** *BF*

	$\lvert V\rvert$	average degree			
		2.5	5.0	10.0	20.0
random	124	1000	1000	999(2)	978(7)
	250	1000	1000	1000	1000
	500	1000	1000	1000	1000
	1000	1000	1000	1000	1000
		5	10	20	40
uniform	250			315(89)	
geometric	500	1000	1000	966(4)	313(60)
	1000	1000	1000	1000	885(7)
			9+16		
planted	250		1000		
bisection	500		1000		

Table IV.15: **Average/best partition cost for** *BF*

	$\lvert V\rvert$	average degree			
		2.5	5.0	10.0	20.0
random	124	24.0/ 15	76.6/ 64	193.5/ 179	472.4/ 449
	250	57.0/ 40	142.3/121	392.7/ 364	868.1/ 831
	500	102.8/ 80	285.9/249	699.8/ 653	1833.6/1770
	1000	202.7/170	580.5/532	1523.1/1461	3581.2/3485
		5	10	20	40
uniform	250			221.8/140	
geometric	500	61.1/28	147.3/ 52	347.3/178	694.7/412
	1000	136.3/88	312.8/170	689.2/333	1333.7/737
			9+16		
planted	250		1174.4/1133		
bisection	500		2319.1/2261		

Table IV.16: **Average number of evaluations for** *BS*

	$\lvert V\rvert$	average degree			
		2.5	5.0	10.0	20.0
random	124	68108	77710	86236	94708
	250	492094	607047	687000	747141
	500	3861125	4547625	5069500	5826063
	1000	30974000	35498750	40399750	45676000
		5	10	20	40
uniform	250			941547	
geometric	500	5672875	6608188	7427313	7863250
	1000	43618250	52001750	58177000	62611250
			9+16		
planted	250		772516		
bisection	500		6033000		

Table IV.17: **Number of local minima for** *BS*

	$\|V\|$	average degree			
		2.5	5.0	10.0	20.0
random	124	1000	1000	1000	992(3)
	250	1000	1000	1000	1000
	500	1000	1000	1000	1000
	1000	1000	1000	1000	1000
		5	10	20	40
uniform	250			355(65)	
geometric	500	1000	1000	938(5)	213(138)
	1000	1000	1000	1000	782(13)
			9+16		
planted	250		1000		
bisection	500		1000		

Table IV.18: **Average/best partition cost for** *BS*

	$\|V\|$	average degree			
		2.5	5.0	10.0	20.0
random	124	24.3/ 15	79.5/ 65	196.7/ 179	475.9/ 449
	250	59.4/ 45	147.1/122	400.8/ 369	876.7/ 840
	500	106.2/ 85	298.0/258	717.8/ 672	1856.1/1792
	1000	214.6/188	610.0/570	1567.1/1498	3634.8/3530
		5	10	20	40
uniform	250			222.7/140	
geometric	500	68.6/ 25	153.5/ 49	320.4/178	599.4/412
	1000	158.5/110	323.8/172	641.9/287	1163.1/737
			9+16		
planted	250		1184.2/1125		
bisection	500		2341.9/2269		

the geometric graphs, for which the longest searches are observed for the degree ten graphs. This effect appears to be robust for this LS method, holding for graphs of size 500 and 1,000, but it is not observed for any other LS method.

More direct evidence regarding the number of local basins can also be seen in the tables. We see that for almost every graph, no two local minima (out of the 1,000 found) were the same. This implies a very large number of local basins, as not even a chance occurrence resulted in seeing the same basin more than once.[2] It is not clear to what extent this reflects complex structure of the search space. As discussed in Section IV.A.2, there can be large equivalence classes of distinct local minima which are not different in any meaningful way (for example nodes of degree zero can be swapped with no effect on partition cost).

For five of the graphs, several duplicate local minima were found by at least one of the LS methods. This implies either that these graphs have fewer basins, or that they have some much larger basins. All of these graphs have high average degree, which is consistent with the inference drawn above about high degree leading to larger and fewer basins.

A final observation to make about the graphs is that the average cost of the local minima increases with the number of nodes and the average degree. This is to be expected, as larger and more connected graphs will require more edges to be cut by any partition.

Comparison of Local Search Methods

Tables IV.19 and IV.20 redisplay some of the information in Section IV.B.2 to facilitate comparison of the local search methods. Table IV.19 shows the average partition cost of the local minima found by LS and Table IV.20 shows the average number of evaluations per LS.

The data shown are for a subset of the graphs examined in Section IV.B.2. This subset is used for the comprehensive empirical study of EA+LS in Section IV.D.

[2]A quick calculation shows that if there are 700,000 equal size basins, the chance that there will be no duplicates after choosing 1,000 at random is approximately 50%.

The subset provides instances of various sizes (124, 250, and 500 nodes) and types (random, uniform geometric, and planted bisection graphs) for a given average degree. All but one of the graphs has expected degree between 20 and 25. We believe these to be more difficult than the smaller degree graphs as there are fewer free nodes (recall Section IV.A.2). Finally, one graph is included which has expected degree 40, to gauge the effect that this has on algorithm performance.

Table IV.19: **Average partition cost**

	BS	**BF**	**US**	**UF**	**HS**	**HF**
r0124.20	475.9	472.3	475.2	474.1	478.9	479.0
r0250.20	876.7	868.1	875.9	872.1	878.0	875.9
r0500.20	1856.1	1833.6	1854.1	1843.5	1856.1	1844.6
g0250.20	320.3	347.3	320.1	386.5	343.1	466.1
g0500.20	641.9	689.2	642.1	821.0	663.5	910.0
g0500.40	1163.0	1333.6	1146.7	1479.0	1222.1	1860.2
p0250.9+16	1184.2	1174.4	1184.5	1179.3	1186.3	1183.8
p0500.9+16	2341.8	2319.1	2341.6	2330.7	2343.2	2332.1

There are several observations to make from Table IV.19. First, the relative effectiveness of the LS methods depends on the type of graph being searched. For almost all random and planted bisection graphs, and all neighborhood structures, first-improve LS finds slightly better solutions than steepest descent. The differences are small but consistent. That steepest descent often does worse may seen counterintuitive. The explanation is that this greedy method can quickly get "stuck" in a nearby basin, whereas the first-improve method has more freedom to roam over multiple basins, due to its stochasticity.

For the geometric graphs, in contrast to the above, steepest descent finds markedly better solutions. For these highly structured graphs, there is a real benefit to making as much progress as possible with each step. As we will see below, however, these greatly superior solutions come at the expense of speed.

Comparing neighborhood structures, the results again depend on the type of graph being searched. For the random and planted bisection graphs, there is little difference between the three neighborhoods. The best average solutions are found with the balanced first-improve method, but these are at most 1% better than the solutions found with the other neighborhoods using first-improve. Neighborhood structure makes more of a difference on the geometric graphs. On these graphs, using steepest descent, the balanced and unbalanced neighborhoods have similar solution quality, with half-balanced somewhat worse (up to 7%). Using the first-improve method, there is a much bigger difference between the neighborhoods, with balanced being the best and half-balanced by far the worst.

Table IV.20: **Average LS length:** For comparison, a single step of LS using the balanced neighborhood requires 3844, 15,625, or 62,500 evaluations for a graph of size 124, 250, or 500, respectively.

	BS	**BF**	**US**	**UF**	**HS**	**HF**
r0124.20	95k	9516	6101	1213	4264	824
r0250.20	747k	35939	23833	2726	17654	2283
r0500.20	5826k	138000	93570	6646	69762	6057
g0250.20	7427k	153553	117875	6221	85197	3736
g0500.20	58177k	578481	463704	14796	342511	9826
g0500.40	62611k	683845	498826	19866	360123	9636
p0250.9+16	773k	38598	24029	2788	17996	2259
p0500.9+16	1738k	144486	96218	6956	72015	6234

Two important trends regarding LS *length* can be seen in Table IV.20. The first is that for any neighborhood structure, the first-improve method is much quicker than steepest descent. Depending on the graph and neighborhood, steepest descent uses anywhere from five to 90 times as many evaluations as first-improve. This points to how much time is saved by consistently finding quick improvements in the early and middle stages of local search. In the final stages of LS, when the search is near a local minimum, improving neighbors are hard to find and there is not as much difference between steepest descent and first-improve.

The other trend to notice is the enormous amount of time spent when using the balanced neighborhood. With its quadratic neighborhood size, it is necessarily much slower than the other methods. Even when using the first-improve method, in which the full neighborhood does not have to be searched every time, the balanced neighborhoods still suffer. This has an easy explanation. Even with first-improve, the entire neighborhood has to be searched at least once, namely on the last step of LS before returning. This full search is necessary to verify that the search has reached a local minimum. Even this one search of the complete neighborhood takes roughly as many evaluations as an entire local search using one of the other neighborhoods.

The balanced neighborhood's slowness has obvious consequences for our intended use of LS as part of an EA. On the geometric graphs, the only graphs for which the balanced neighborhood gives substantially better solutions, the running time ranges from 40 to over 100 times as slow as the half-balanced method, depending on the graph and whether steepest descent or first-improve is used. This means that an EA could sample many regions of the global search space and do half-balanced LS on these samples in the time it would take to do balanced LS on a single partition. If our intuitions about EA+LS are correct, it is much more important to gather many samples and quickly refine them than to spend enormous effort refining a single point. For example, in the most extreme case, comparing balanced steepest descent with half-balanced first-improve on the graph g0500.40, the latter is almost 6,500 times faster, and would allow a complete EA+LS run to finish before one LS of the former.

The comparison between the unbalanced and half-balanced neighborhoods is more interesting. For all graphs and for both steepest descent and first-improve, the half-balanced neighborhood is quicker. The unbalanced neighborhood uses from 10 to 106% more evaluations. This difference is substantial, but the tradeoff is solution quality. The unbalanced neighborhood always gives better solutions, particularly on the geometric graphs. The biggest disparity between unbalanced and half-balanced, both in terms of solution quality and running time, occurs on the geometric graphs using first-improve. This interesting tradeoff is explored in Section IV.D.2 in the context of an EA.

In summary, there is little difference in solution quality between the various LS methods for random and planted bisection graphs, so the speed of LS alone may be expected to determine its usefulness. On the geometric graphs, the LS method makes a big difference in solution quality, with steepest descent clearly superior, and with the half-balanced neighborhood somewhat worse than the other neighborhoods. For all graphs, the balanced neighborhood is too slow to be useful in comparison to the other methods, and the half-balanced is substantially faster than unbalanced. Also in all cases, first-improve is quicker than steepest descent. For the random and planted bisection graphs, these observations suggest that half-balanced first-improve will be most successful as part of an EA+LS algorithm. For the geometric graphs, there are tradeoffs between speed and quality, and there is no clear recommendation.

IV.C Evolutionary Algorithm

As a baseline for evaluating the usefulness of local search, we first examine an EA *without* LS. In this section the EA is compared to random sampling, Monte Carlo local search, and finally the EA+LS. We also examine the effectiveness of crossover, as a prelude to Section IV.D.4, in which a corresponding set of experiments is performed in the context of LS.

IV.C.1 Baseline Results

As a first check of the EA's effectiveness, we compare it to *Monte Carlo* (MC) search, which is simply unbiased random sampling (note that this is not the same as Monte Carlo local search, discussed in the previous section, as no local search is done). The EA used in the experiments is fairly typical. It is generational, with an expected mutation size of five node swaps per solution. All other algorithmic parameters are set to their standard values (see Section IV.D).

Figure IV.4 includes a comparison of the EA and Monte Carlo search. The curves shown for MC are simply plots of the best solution found as a function of the number of solutions considered. In other words, we repeatedly generate solutions uniformly at random over the search space, and keep a record of each time a solution is found which is better than any found previously. Note that each solution is drawn independently, so the curves simply reflect the probability of finding solutions of various fitness. Nevertheless, this presentation allows a direct comparison with the EA in terms of the speed and quality of search. Note that the MC curve has no error bars, as only a single "run" is done.

We see that on all graphs the EA does substantially better than MC, beginning with the earliest stages. Even when MC is allowed ten times as many evaluations as the EA, its solutions are substantially worse. The EA is clearly effective at biasing its sampling towards good regions of the space.

The EA is also compared to Monte Carlo local search using half-balanced first-improve LS. The MCLS plots are generated in a similar fashion to those for MC; local search is run to completion on a sequence of randomly generated starting solutions. A record is kept of each time a solution is encountered which is better than any seen previously. Note that the local searches are done in series: one LS runs to completion before the next one begins.

We see from the figure that MCLS finds much better solutions than the EA, and that it finds them very quickly. In fact, the *average* local minimum has better cost (recall Table IV.3) than the final EA solutions, even though the average local search length is less than 1% of the time spent by the EA. This illustrates the power of LS for graph bisection; even though the EA's global search is very effective when compared to random search, it is completely inadequate in comparison to simply doing local search from a few random starting points.

Since local search itself it so powerful, the use of a population-based sampling algorithm (EA) needs to be justified. We compare MCLS to the generational EA+LS described in Section IV.D.1, which finds the best solutions of all the EA+LS variants considered in this dissertation. Figure IV.5 displays the performance of this algorithm against MCLS. We see that on the random and planted bisection graphs, there is indeed a benefit to having the EA choose the starting points for LS. The difference between MCLS and EA+LS is significant on each of these graphs. There is no significant difference on the geometric graphs, though on average the EA+LS

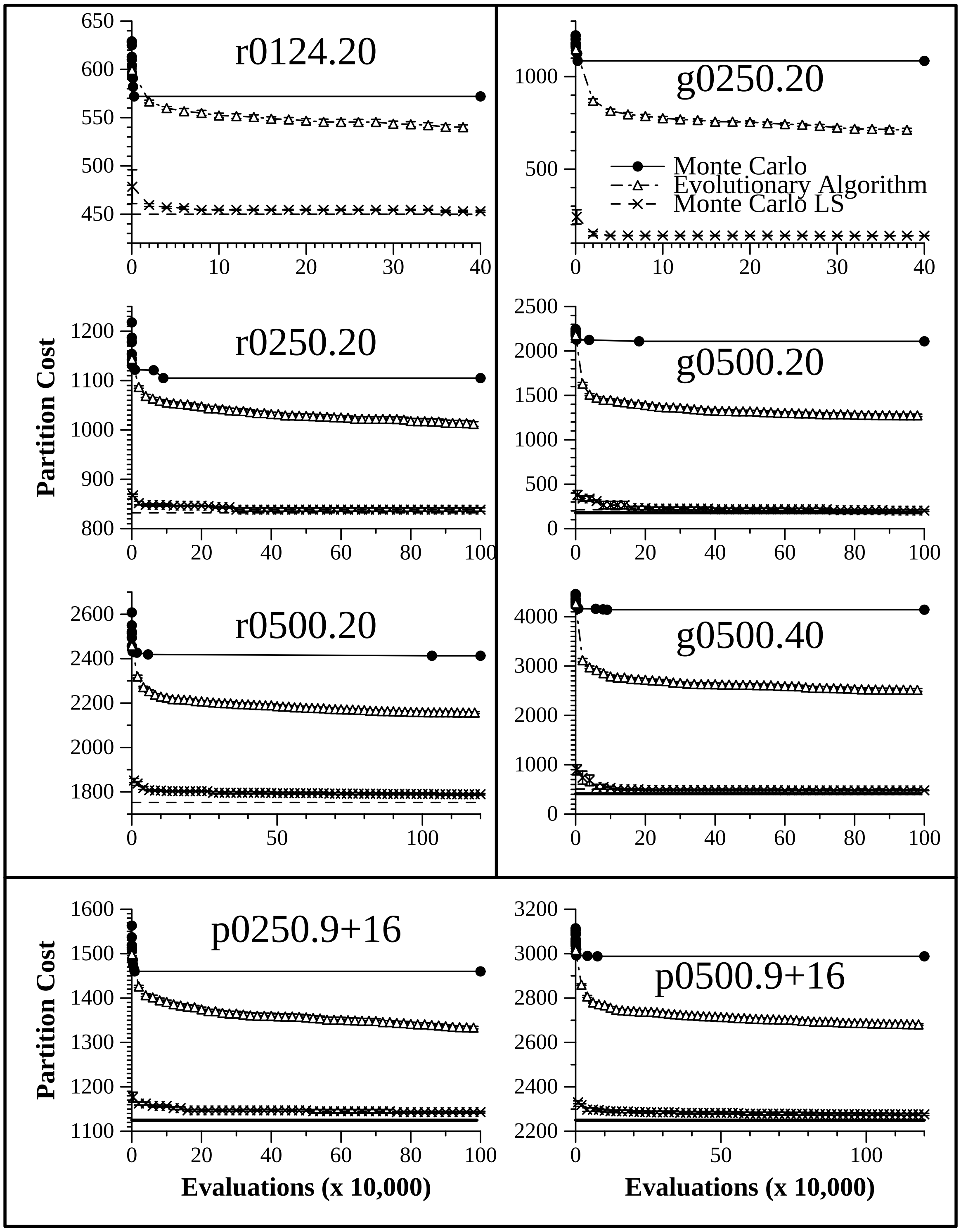

Figure IV.4: Comparison of Monte Carlo, EA, and Monte Carlo LS

performed as well as or better than MCLS on all three.

In summary, the EA effectively biases its sampling towards good regions of the search space, as compared to random sampling. Local search, however, is so powerful that doing even a single LS from a random starting point will probably result in a superior solution, in a fraction of the time used by the EA. Nevertheless, the EA's population-based global search is superior to random sampling in choosing favorable starting points for LS.

A note about MCLS plots The data for the MCLS plots in this section are taken from the experiments described in Section IV.B.2, in which local search was performed on 1000 random initial starting points. In order to generate the curves, these independent local searches are grouped to form several "runs" of the same length as the EA+LS. For example, on r0124.20 the EA+LS is allowed to run for 400,000 evaluations. A single HF local search on this graph uses 824 evaluation on average. Hence, 485 $(400,000/824)$ complete local searches are equivalent to an EA+LS run. Therefore, we consider the first 485 local searches to be a single MCLS run, the next 485 to be a another, and so on. Since only 1000 local searches were initially performed, we get only two runs for this graph. By grouping the data in this way, we get a fair comparison between the methods (as they use the same number of evaluations) and an improved ability to do statistical comparisons, as there are multiple MCLS runs for each graph.

IV.C.2 Effect of Crossover

The general purpose of the crossover operator is to recombine useful parts of distinct genotypes to create a solution which has the best of both parents. The ability of crossover to do this depends of course on the problem at hand, and also on the way solutions are represented and the specifics of the crossover operator. Generic operators may be suitable for many problems, if there is a straightforward binary representation. For example, the k-SAT problem has an obvious representation, a

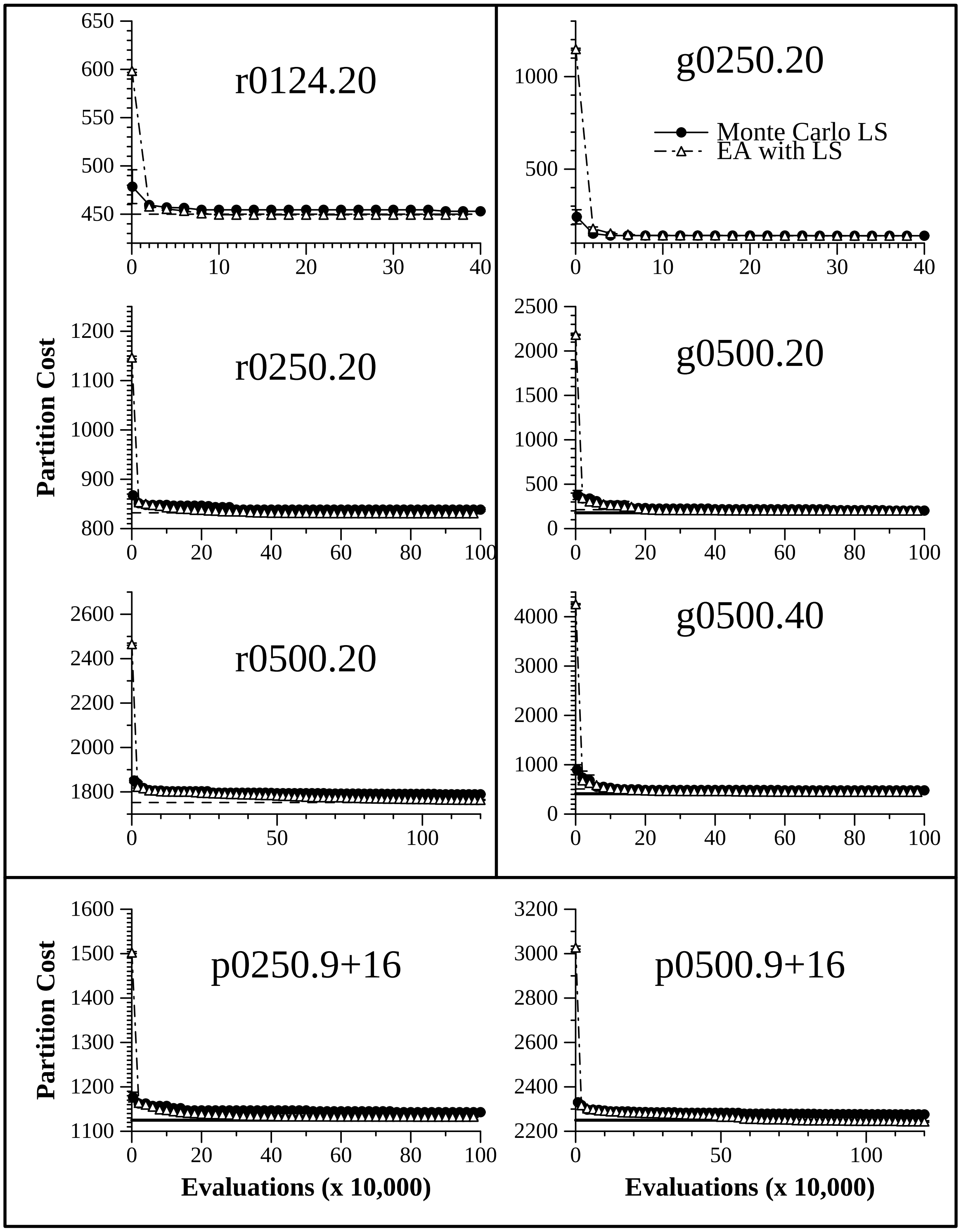

Figure IV.5: Comparison of Monte Carlo LS and the EA with LS

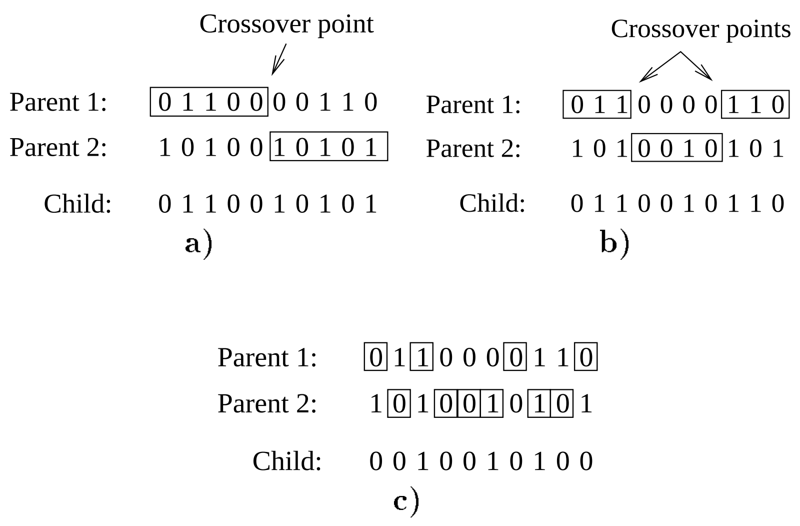
Crossover point
Parent 1: 0 1 1 0 0 0 0 1 1 0
Parent 2: 1 0 1 0 0 1 0 1 0 1
Child: 0 1 1 0 0 1 0 1 0 1
a)
Crossover points
Parent 1: 0 1 1 0 0 0 0 1 1 0
Parent 2: 1 0 1 0 0 1 0 1 0 1
Child: 0 1 1 0 0 1 0 1 1 0
b)
Parent 1: 0 1 1 0 0 0 0 1 1 0
Parent 2: 1 0 1 0 0 1 0 1 0 1
Child: 0 0 1 0 0 1 0 1 0 0
c)

Figure IV.6: **Three standard crossover operators:** a) One-point crossover, b) two-point crossover, c) uniform crossover. The boxes indicate which parent each bit is chosen from.

single bit for each variable indicating its truth value.

Standard crossover operators such as one-point, two-point, and uniform crossover simply assign each bit in the child to be the same as the corresponding bit in one of the parents (see Figure IV.6). Specifically, one-point crossover randomly chooses a "crossover point" on the genotype: all bits before that point are copied from one of the parents and all bits after the point are copied from the other parent. Two-point crossover is similar, but there are two crossover points. The bits before the first point and after the second point are copied from one parent, and the bits between the two points are copied from the other. One- and two- point crossover respect locality on the genotype, so that bits which are near each other are more likely to be chosen from the same parent than bits which are far apart. Finally, uniform crossover chooses between the parents independently for each bit, so that there is no linkage between positions on the genotype.

Graph partitioning allows a straightforward binary representation. Specifically, the genotype has one bit for each node, specifying which side of the partition that node is on. While simple, this representation has a few drawbacks. First, any given partition has two distinct encodings. This can be problematic for crossover, as two parents may be quite similar to each other but have very different encodings. This may cause crossover to disrupt parts of the solution on which the parents agree, because the corresponding sections of the genotype are different. We will examine ways to deal with this problem, including "normalizing" representations before recombination so that they are more similar.

A second problem with this representation is that the common crossover operators described above often produce invalid partitions. Valid partitions must have the same number of nodes on each side of the partition, but this property is not preserved by any of the operators described. For all graph partitioning experiments in this dissertation we use the following crossover operator: for each bit on which the parents have the same value, this value is copied to the child. The settings for the remaining bits are chosen uniformly at random from all possible settings which have the required number of 0s and 1s. Hence, crossover always produces a valid partition. Furthermore, it preserves any genetic information that both parents possess.

This operator is a special case of the RAR_w subset recombination operator described by Radcliffe [70, 69]; in particular it is equivalent to RAR_∞. In preliminary experiments we tried RAR_w for various values of w and found RAR_∞ to be robust across all graph instances. Note also this operator ignores locality on the genotype, i.e. nodes which are adjacent on the genotype are no more or less likely to be copied from the same parent than nodes which are far apart. This is reasonable for random graphs since the order of nodes on the genotype does not reflect any prior expectations about which nodes should be transferred together. For geometric graphs there may be ways to arrange the genotype so as to exploit the structure, e.g. putting geometrically close nodes near other on the genotype. We make no such attempt here, instead using a random ordering of nodes.

Symmetric Normalization of Partitions

As discussed above, the existence of duplicate representations for each partition may be problematic for crossover. In order to alleviate this problem, we try various methods of normalizing the partitions' representations before performing crossover. Intuitively, if two parents are similar we want their representations to be similar also. Hence, an obvious procedure is to examine each pair of parents before applying crossover and to invert (flip all the bits of) one of them if this brings their representations closer together. Here distance is defined to be Hamming distance. We call this procedure *pairwise Hamming* normalization. Note that under this method the actual parity of the representation used for a partition in the population is irrelevant, as the bitstrings are always normalized before interacting with each other.

Several alternative normalization procedures are compared to the pairwise Hamming method. The *population-wide Hamming* method normalizes the entire population with respect to the current best solution in each generation. The *population-wide fixed node* method selects a single reference node in the graph prior to the start of the EA run. Each generation, the entire population is normalized with respect to this node, i.e. the bitstrings are set so that the bit corresponding to this node is 0 in every genotype. The motivation for this is that it eliminates the duplicate representation problem; the symmetry is broken by fixing the reference node. The *population-wide random node* method is similar, but the reference node is selected randomly every generation. The *random* method simply inverts each parent with probability 0.5. This is used as a test to see if normalization matters at all. Finally, the *null* method does no normalization. For all methods, any normalization is applied before crossover.

Experiments are performed comparing the various normalization methods. A generational EA without local search is used, with a population size of 150. The per-node mutation probability is 0.002. Since the EA is generational, any normalization that occurs does not directly affect succeeding generations. Its only affect is by modifying the operands of crossover. Table IV.21 displays the relative performance of each of these methods on the 24 random and geometric graphs from [44]. Each number in the table represents the best solution found after 200 generations, averaged over ten runs. To facilitate comparison across different graphs, the numbers are scaled with respect to the pairwise Hamming method, so that a rating of less than 1.0 indicates better performance than pairwise Hamming. We see that the there is little difference between the pairwise Hamming, population-wide Hamming, and null methods. The population-wide random node and random methods both do substantially worse on all graphs. The population-wide fixed node method is comparable to the best methods on the random graphs and is somewhat worse on the geometric graphs, though still much better than the worst methods.

It is not surprising that the random method does so poorly; under this method similar parents will have dissimilar representations as often as not. Given this result, then, it is somewhat surprising that the null method does as well as all the others. If it is detrimental to make the representations dissimilar (a la the random method), then why isn't it beneficial to always make them similar? One possibility is that the population automatically converges onto one of the two representational conventions. In this case all the bitstrings would be more similar than dissimilar, and normalization would have no effect.

Also curious is the fact that the population-wide random node method does almost as poorly as the random method. It may be the case that choosing a random node as a reference each generation has the same effect as random normalization. As an extreme example, consider what happens if the chosen reference node has degree zero. Then in each genotype it is equally likely to be 0 or 1, regardless of the rest of the solution. Hence, setting the bitstring so that this node's bit is 0 is effectively the same as randomly deciding whether or not to invert it. Note that the situation is different for the population-wide *fixed* node method. In this case, even if the fixed node has degree zero, there is a fixed representational choice which allows the population to converge.

Crossover vs. Macromutation

A sometimes controversial issue is whether a crossover (or recombination) operator is useful as part of an EA [41, 46, 23]. Proponents of crossover argue that

Table IV.21: **Comparison of normalization methods:** The relative effectiveness of various normalization methods on 24 graphs. Each entry represents the fitness of the final solution found by an EA, averaged over ten runs. All figures are normalized with respect to the pairwise Hamming method, which be definition has value 1.0.

graph	Pairwise Hamming	Pop-wide Hamming	Pop-wide fixed node	Pop-wide random node	random	null
r0124.03	1.00	0.96	0.94	2.26	2.28	0.88
r0124.05	1.00	1.06	1.05	1.62	1.66	1.01
r0124.10	1.00	1.01	1.01	1.33	1.33	1.02
r0124.20	1.00	0.99	1.00	1.16	1.18	0.99
r0250.03	1.00	1.02	1.03	2.55	2.55	0.97
r0250.05	1.00	0.99	1.02	1.75	1.78	0.97
r0250.10	1.00	1.01	1.02	1.41	1.42	1.00
r0250.20	1.00	1.00	1.01	1.24	1.24	1.01
r0500.03	1.00	0.97	1.14	2.47	2.48	0.96
r0500.05	1.00	1.00	1.07	1.76	1.78	1.01
r0500.10	1.00	0.99	1.03	1.41	1.42	0.98
r0500.20	1.00	0.99	1.02	1.24	1.24	0.99
r1000.03	1.00	0.99	1.15	1.96	1.98	0.97
r1000.05	1.00	0.99	1.08	1.52	1.53	0.99
r1000.10	1.00	1.00	1.05	1.32	1.32	1.00
r1000.20	1.00	1.00	1.02	1.19	1.20	1.00
avg. random	1.00	1.00	1.04	1.64	1.65	0.99
g0500.05	1.00	1.05	1.35	5.39	5.49	1.05
g0500.10	1.00	1.08	1.39	4.65	4.77	0.96
g0500.20	1.00	1.03	1.18	3.99	4.07	1.11
g0500.40	1.00	0.95	1.17	3.62	3.71	0.84
g1000.05	1.00	1.00	1.32	2.88	2.94	1.04
g1000.10	1.00	1.00	1.25	2.60	2.64	0.97
g1000.20	1.00	1.03	1.26	2.53	2.57	1.07
g1000.40	1.00	0.98	1.21	2.46	2.49	0.96
avg. geometric	1.00	1.01	1.27	3.51	3.59	1.00

it is crucial whenever the representation of the problem at hand allows “building blocks.” Roughly, a building block is a schemata which represents a good solution to a subproblem of the problem at hand. For example, in graph partitioning a building block might be a part of the genotype which specifies that some group of nodes is on the same side of the partition. If these nodes form a small clique, then partitions containing this building block will have higher fitness than those without it, on average. The main idea is that if there are several such building blocks, they can be discovered independently (through the normal EA processes of variation and selection) and then combined with crossover. For a more thorough discussion of building blocks and schemata see [41].

A test that is commonly done to evaluate crossover’s effectiveness is to compare an EA with crossover to one without. When crossover is not used, new solutions are generated by simply cloning one parent and applying mutation. If an EA works as well without crossover as with, then crossover is not combining building blocks. Note that if using crossover does help, it does not necessarily follow that it is combining building blocks. It could simply be the case that the variation generated by crossover in the population is beneficial. In this case, it is rather like a large mutation operator, and it is said to produce *macromutations*.

The possibility that crossover benefits search by simply allowing macromutations leads to another test, developed by Jones [46], in which “random” (or “headless chicken”) crossover is used. For random crossover, one parent is selected from the population in the usual way, but the second parent is randomly generated. The crossover operator is then applied to produce a new solution, and the randomly generated parent is discarded. In this way the benefit of macromutation is examined in the absence of any genetic recombination. If an EA with random crossover performs as well as one with normal crossover, this implies that crossover’s benefit derives from its macromutations and not from its combining of building blocks.

Figure IV.7 shows the results of EA runs using the three crossover methods on eight graphs. These runs are performed using a steady-state EA with an expected mutation size of five node swaps per solution. We see that in all cases, standard crossover does substantially better throughout the runs than either alternative. Random crossover is especially poor, improving little after the very early stages. Since macromutations are so detrimental, they cannot be the source of the benefit for normal crossover. This suggests that crossover does indeed combine building blocks from disparate solutions. Section IV.D.4 describes a similar experiment for an EA which used local search.

IV.D Evolutionary Algorithm with Local Search: Mechanisms of LS Interaction

In this section we explore the effect of varying several algorithmic parameters of the EA+LS hybrid. For most experiments we employ a new variant of the EA+LS, which allows us fine control over the various aspects of the global/local interaction. Figure IV.8 outlines the algorithm. This is a steady-state algorithm in which LS applied to every individual upon creation. Only a small amount of LS (a "sniff") is used; this may give a sense of how much potential there is for improvement with further local search. Each time a solution is created (and given a LS sniff), other solutions may be selected to receive additional LS. Various methods may be used to select which individuals get LS; for example it may be based on fitness or on an estimate of the potential for future improvement. The unusual feature of this algorithm is that it uses only small amounts of LS at a time, and that it allows various mechanisms for selecting how the LS effort is allocated to the population. The goal is to maintain a population of solutions which have each undergone *some* amount of LS, so that useful comparisons can be made between members. In contrast, algorithms in which a fraction of the population undergoes LS to completion may suffer from the problem that the best solutions are simply those which were chosen for LS, and so selection may be skewed.

More generally, this new algorithm integrates global and local search more

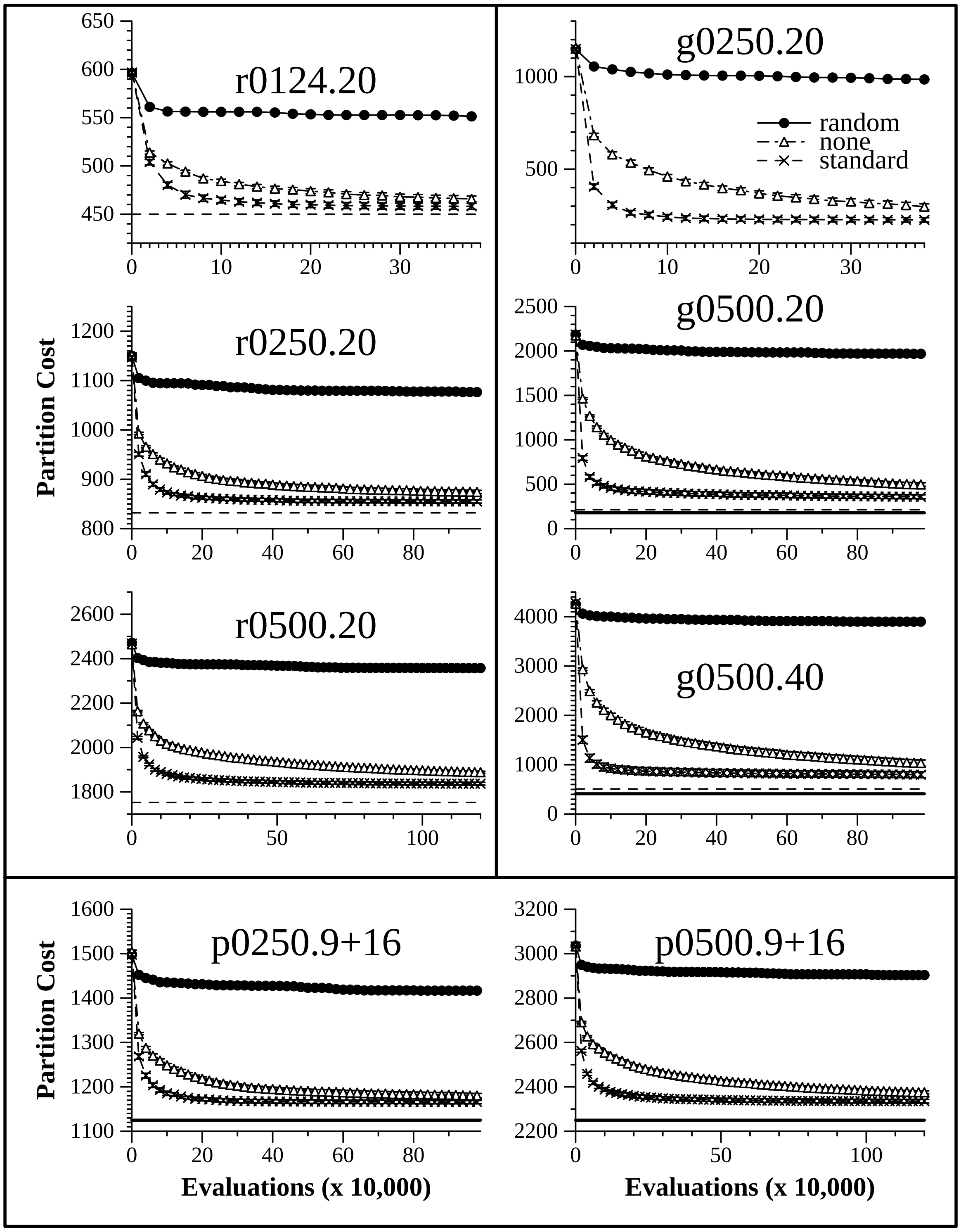

Figure IV.7: Comparison of crossover methods

```
Randomly generate an initial population of size M
Do a LS sniff on each member
Repeat
    Pick two solutions, biased by fitness
    Create new through recombination and mutation
    Do LS sniff on new
    Replace the worst member of the population with new
    Select (LS rate) additional members to do
            (LS increment) additional LS
Until good enough solution found, or out of time
```

Figure IV.8: **Steady-State EA with Short LS:** This algorithm is used for most of the experiments in this section.

tightly than previous versions of EA+LS (such as the generational method in Section IV.D.1). Typical EA+LS hybrids proceed in alternating stages of global and local search. First the EA produces a new population, then local search is performed. The specific state of local search generally is not kept from one generation to the next, though LS results do influence selection of individuals and, in the case of Lamarckian evolution, the genotypes themselves.

This interleaving of the global and local search phases allows the two to influence each other; e.g. the EA chooses good starting points, and LS provides an accurate representation of that region of the domain. This scheme, however, does not take full advantage of the possibilities for interaction between global and local search. The new algorithm interleaves global and local search at a finer granularity. Selection and reproduction can occur *during* a local search (i.e. in between successive stages of LS on a single solution), and the amount of LS done on a solution can depend on the success of whatever LS has already been done on it. Hence, population-wide statistics can direct the use of LS at a fine level, for instance by determining exactly how much

LS to apply to each solution based on fitness, nearness to other population members, or other features. Furthermore, the state of LS is maintained in the population, unlike in the generational case, and can directly influence future selection.

An expanded version of the algorithm is shown in Figure IV.9, with the various algorithmic parameters highlighted. Next to each parameter, a list of possible instantiations is given. In fact, these values are all used in one or more of the experiments in this section.

The canonical parameter settings we use to compare with all variations are as follows: the length of LS sniff as well as LS increment is ten evaluations. The number of extra individuals chosen for LS each time through the loop (LS rate) is two. Hence, thirty evaluations of LS are used for each evaluation of "global" search (new individual generated by selection genetic operators). The extra individuals are chosen randomly. The half-balanced first-improve LS method is used unless otherwise stated, and evolution is Lamarckian. We use we use standard RAR_∞ crossover. Finally, for most experiments a mutation size of $5/N$ (an expected five node swaps per genotype) is used, though Sections IV.D.1 and IV.D.2 use a fixed per-node mutation probability of 0.002. Except where explicitly varied, these parameter values are in use throughout this section.

The remainder of this section is broken down into investigations of how these various parameters affect search performance. Section IV.D.1 first compares our algorithm with a generational EA+LS and a more traditional steady-state EA+LS. Sections IV.D.3 and IV.D.4 examine the roles of mutation and crossover when LS is used. Section IV.D.2 examines the effect of the LS method used. Section IV.D.5 examines the effect of using various amounts of local search and different LS lengths, as well as omitting the LS sniff given to every individual when it is created. Finally, Section IV.D.6 looks at methods for choosing which solutions get additional LS.

There are a few algorithmic details which are constant across all runs in this section: the population size is 200; the crossover operator used is RAR_∞, and no normalization of partitions is done beforehand (see Section IV.C.2 for an explanation

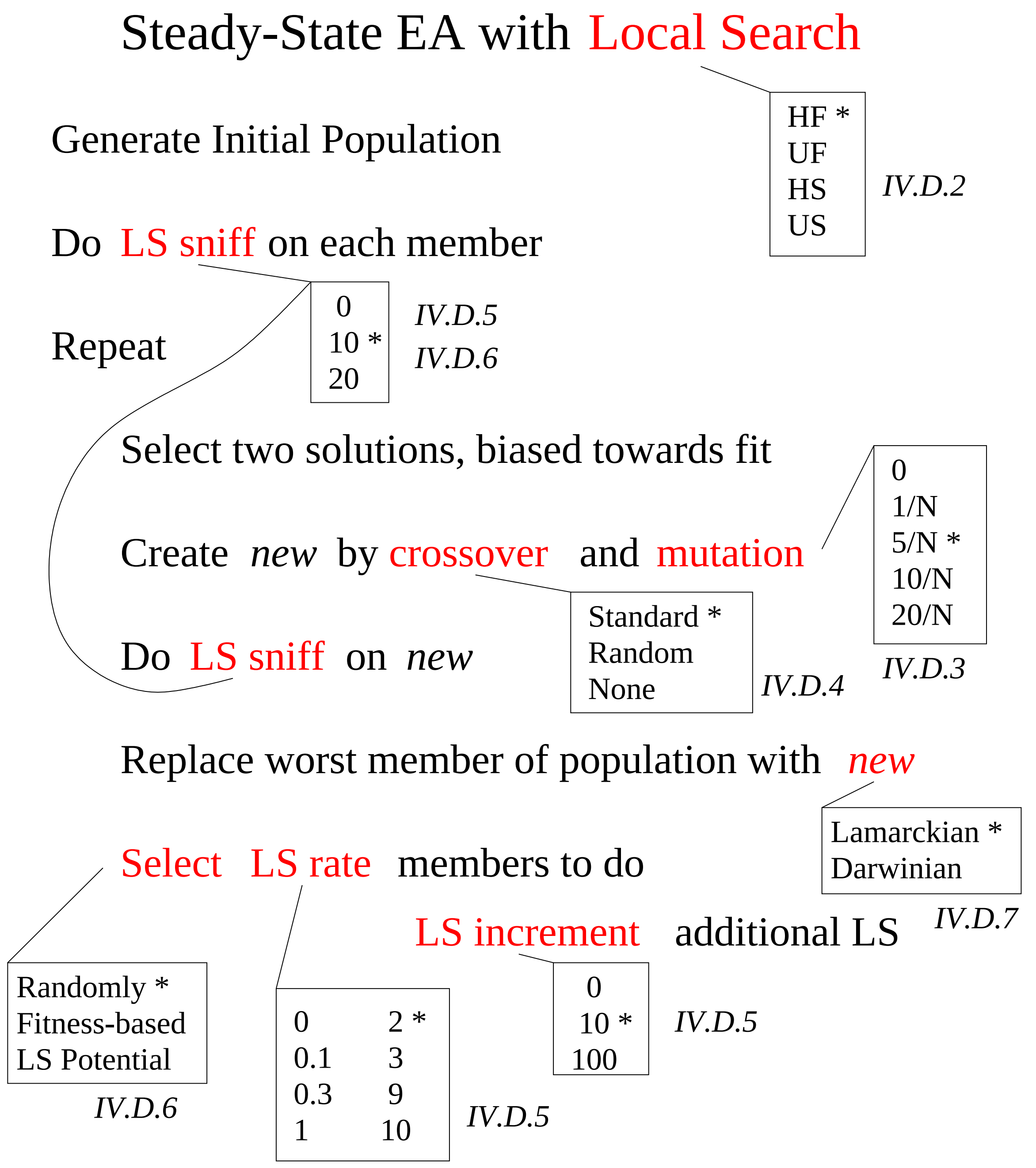
Steady-State EA with Local Search
HF * UF HS US
IV.D.2
Generate Initial Population
Do LS sniff on each member
0 10 * 20
IV.D.5
IV.D.6
Repeat
Select two solutions, biased towards fit
Create new by crossover and mutation
0 1/N 5/N * 10/N 20/N
IV.D.3
Standard * Random None
IV.D.4
Do LS sniff on new
Replace worst member of population with new
Lamarckian * Darwinian
IV.D.7
Select LS rate members to do
LS increment additional LS
Randomly * Fitness-based LS Potential
IV.D.6
0 0.1 0.3 1 2 * 3 9 10
IV.D.5
0 10 * 100
IV.D.5

Figure IV.9: **Expanded Steady-State with Short LS algorithm:** The various algorithmic parameter values used in this section are listed in boxes next to the appropriate variables, with the default values starred. In italics next to each box are the sections in which that parameter is explored.

of both of these issues); and the selection algorithm used is stochastic universal selection [26], which selects individuals proportionally to their fitness, but attempts to do so in a way which minimizes random influences.

IV.D.1 Generational vs. Steady-State

The EA used for most of our experiments (see Figure IV.8) is somewhat different from the typical EA. One potentially important difference is that we use a steady-state as opposed to a generational replacement scheme. The other differences have to do with how local search is used. Specifically, a typical EA+LS might use complete LS, whereas we use only very small amounts of LS at any one time. To check how our algorithm compares against more traditional methods, we compare it to a fairly typical generational EA+LS and also to a steady-state version which is otherwise the same as the generational method.

The generational EA we use is Lamarckian, employing the half-balanced first-improve LS method. Each generation 5% of the population is chosen at random to receive complete local search (LS runs until completion). There is a per-node mutation probability of 0.002, and simple elitism is used. The same parameters are used for the steady-state variant, with each newly generated solution replacing the worst member of the population, and then undergoing local search with probability 5%. We will refer to this method as the *steady-state with complete LS* method.

The generational and steady-state with complete LS methods are used as a baseline to rate the *steady-state with short LS* algorithm used throughout this chapter. This method allocates LS in a radically different manner: each time a new solution is generated it undergoes a LS sniff of length ten. Then two other solutions are chosen at random from the population and these also undergo ten evaluations each of LS. The state of each solution's LS is stored, so that if it is later selected to receive additional LS this search is continued where it left off previously. Hence LS can eventually reach local optima but will always require several applications to do so. All other algorithmic parameters are the same as for steady-state with complete LS.

Figure IV.10 displays the results of the three methods on eight graph partitioning instances. We see that the generational and steady-state with complete LS methods are virtually identical in performance. In six of the eight graphs there is no statistical difference between the final solutions produced. This reassures us that using a steady-state replacement strategy is not harming the chances of having an effective algorithm. The comparison with steady-state with short LS is not favorable, however. On all graphs this method produces the worst final solutions (the difference with generational is statistically significant on all but the smallest graph). The modifications regarding how LS is allocated are apparently quite detrimental to the search. Nevertheless, because this method allows fine control over the use of LS we will use it as a starting point to investigate how global and local search issues interact. We will see later how the proper settings of the EA parameters allow it to be competitive with the generational method (see especially Section IV.D.3).

As a final observation, note that the difference in performance between steady-state with short LS and the other methods is especially large early in the runs. This is simply due to the fact that LS is unable to run to completion on any solution until several generations have passed. Because short LS is applied to randomly chosen members frequently, we expect that all solutions in the population have usually had roughly the same amount of LS—in early generations, this amount is small in comparison to the length of a complete LS. Contrast this to the complete LS methods, in which approximately 5% of any population in the first generation is at a local minimum. The result is that these methods have high-quality solutions almost from the start.

IV.D.2 Choice of Local Search Method

As discussed in Section III.C.1, the method used for local search can be expected to have a substantial impact on the performance of an EA+LS hybrid. The main concern is the tradeoff between the quickness of local search and the quality of the solutions it produces. Increasing the neighborhood *size*, for example examining all

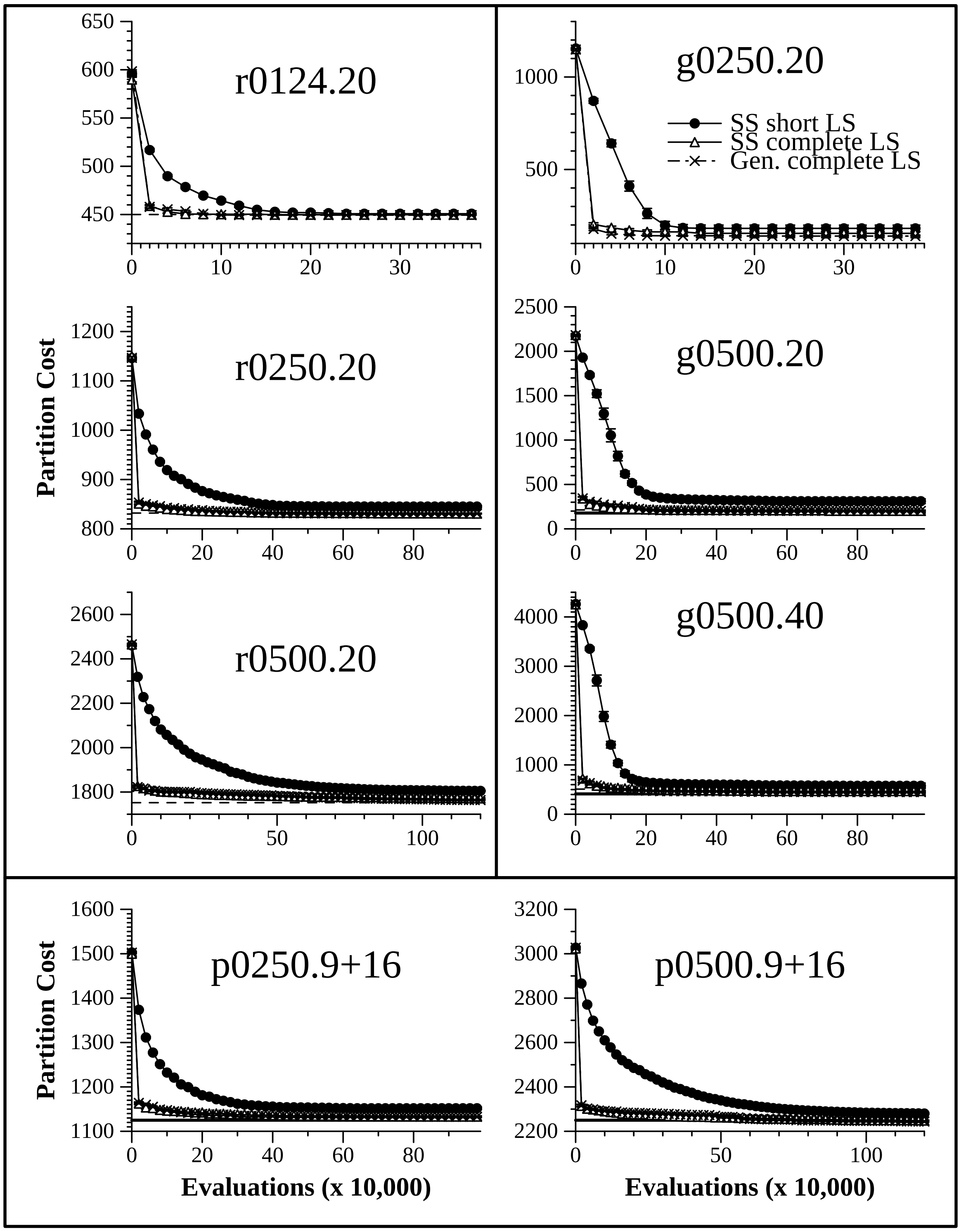

Figure IV.10: Generational vs. Steady-State

pairs of bit flips rather than single bit flips, will result in better solutions. This benefit, however, may not be worth the extra time spent searching the larger neighborhood.

Other aspects of the local search algorithm can also have an impact. Notably, first-improve methods are generally much faster than steepest descent methods for a given neighborhood (recall Section IV.B.2). Despite this, the quality of solutions found by first-improve is often as good as or even better than those found by steepest descent. Clearly, when two LS methods produce roughly comparable solutions, the quicker one can be expected to be a better choice as part of an EA+LS hybrid. Figure IV.11 compares EA+LS using various LS methods on eight instances of graph partitioning. The LS methods used are the four combinations of unbalanced or half-balanced neighborhoods with first-improve or steepest descent moves. Due to its excessive running time, local search with the balanced neighborhood structure was not used as part of an EA+LS.

For each instance, the runs using first-improve LS are superior to those using steepest descent LS. At all stages, the first-improve runs have significantly better solutions. This is certainly at least partially due to fact that we are using very small amounts (ten evaluations) of LS at a time. The steepest descent methods need to examine the entire neighborhood, so n (number of graph nodes) evaluations of LS are required for any change to be made. Since each application of LS uses far fewer than n evaluations, a solution must undergo LS several times to get even a single step of improvement. It is likely that many solutions are replaced from the population before completing this first step, thereby wasting the LS effort that had been applied to them. In contrast, the first-improve methods can make multiple improvements with only ten evaluations.

Figure IV.11 shows virtually no difference between the use of unbalanced and half-balanced neighborhoods. With the exception of the graphs g0250.20 and g0500.40, the two are statistically indistinguishable throughout most of the runs. This observation holds for both first-improve and steepest descent methods. This result seems to point to the tradeoff between solution quality and speed. Despite the

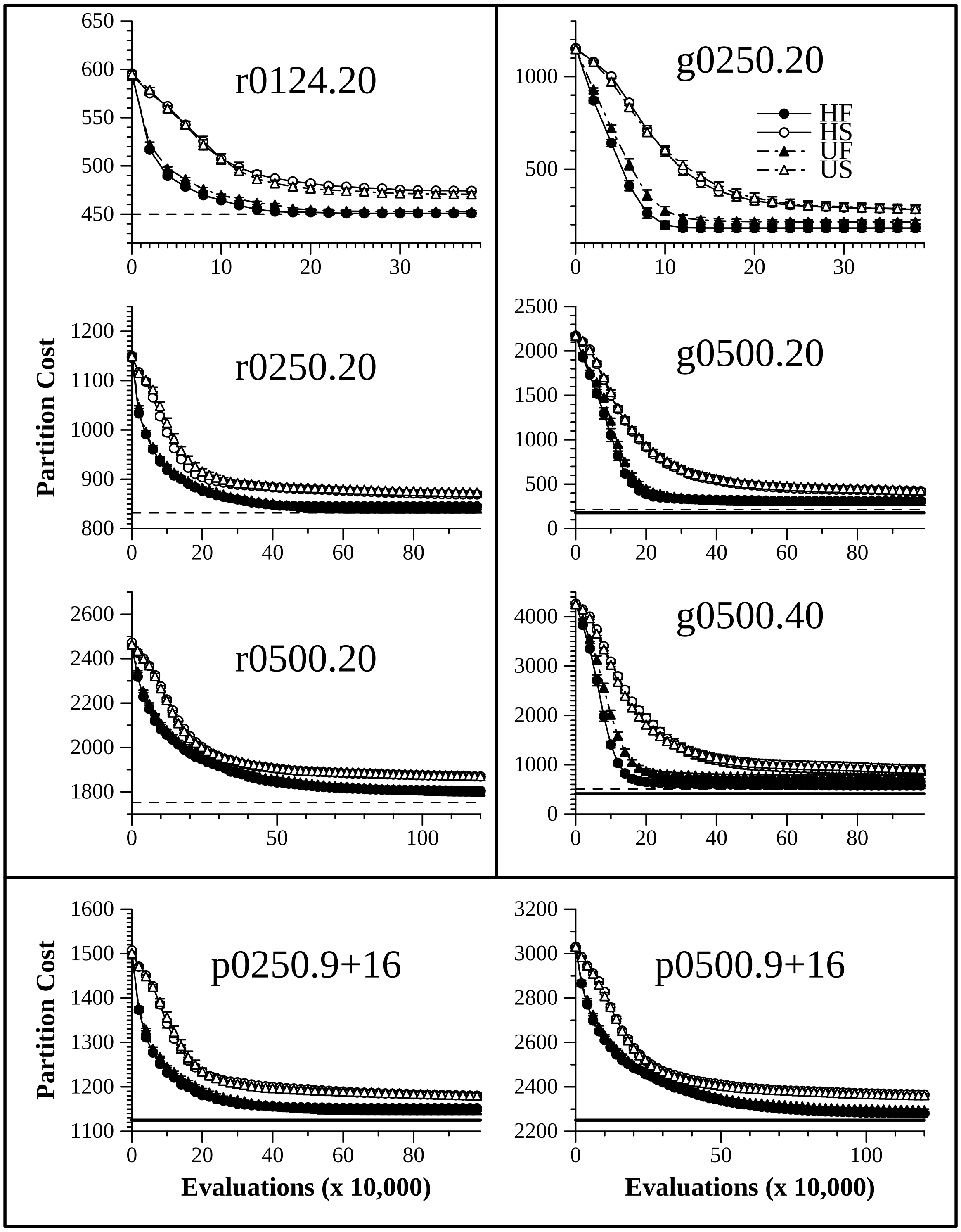

Figure IV.11: Comparison of Local Search methods

fact that the unbalanced LS methods produce better solutions on average (though only slightly better in the case of the random graphs), they are no better than half-balanced for use in an EA+LS. This is apparently due to their longer running time (10-50% longer than half-balanced).

For the remainder of this chapter, the default LS method will be **HF**, as it is as good as the other methods on all graphs, and is statistically better on g0250.20 and g0500.40.

IV.D.3 Role of Mutation

The size of mutations can be expected to play an important role in the effectiveness of the EA+LS hybrid. As discussed in Section III.C.4, mutations which are too small to escape from local basins will have no effect in the later stages of a run, as local search will return such mutated solutions back to the local minimum. As a consequence, mutation's role in preventing convergence and generating new solutions even after convergence will be unfulfilled with such small mutations. Therefore, we expect that in order to be effective in the context of local search, mutation must make larger changes. Specifically, mutations at least as large as local basin sizes will delay convergence and may allow productive search to continue after convergence.

In this section experiments are described which attempt to measure local basin sizes and examine the effectiveness of various mutation sizes.

Measuring Basin Size

In order to determine the size of typical local basins, the following experiment is performed. For each graph instance, 100 initial partitions are chosen at random. Local search is performed to completion on each partition, resulting in 100 local minima. For various mutation sizes, mutants of these local optima are generated, and local search is applied to each mutant. By observing how frequently the mutants are returned to their original local optima, and how this frequency varies with the applied mutation size, we can gauge the size of the basins.

For this experiment, mutation involves swapping 1 to 20 randomly chosen pairs of nodes, depending on the size of the mutation. For each size, 20 mutants were generated. For each mutation size the results of all 100 basins are aggregated, giving a sample size of 2,000 (100 basins times 20 mutants each). Furthermore, this entire experiment is performed for both the HF and UF local search techniques. Figure IV.12 shows the percentage of time that local search returns a mutant to its original local minimum as a function of mutation size, for each graph and LS method.

Small basins The most obvious observation from Figure IV.12 is that all the percentages for 1 or more mutations are low, most less than 25%. This seems to imply that even a single swap is usually enough to escape a local basin. This is somewhat surprising, and makes it appear that any mutation size will allow the benefits discussed above. However, there are a couple of reasons why this conclusion does not follow, and we will argue that this data is merely unhelpful in determining proper mutation size.

First, as discussed in Section IV.A.2, there may be equivalence classes or pseudo-equivalence classes of local minima having the same cost which are one node-swap away from each other. This can happen, for example, if there are nodes of degree zero in the graph instance. Any two such nodes on opposite sides of the partition can be swapped to produce an equivalent partition which is no different in any significant way. In this case, there may technically be a great many very small local basins in a small region of the search space, the only barrier between them being the penalty associated with temporarily unbalancing the graph by moving degree zero nodes.

However, the distinction between these tiny basins is probably not important to the global search. They all have the same cost and lie in the same small region of the space, and can be treated interchangeably, as if they form one large local basin. A mutation operator which merely moves between them is unlikely to be very beneficial. Mutations need to be large enough to escape this "large-scale" basin, and to discover "non-equivalent" basins. In this sense Figure IV.12 is somewhat unhelpful

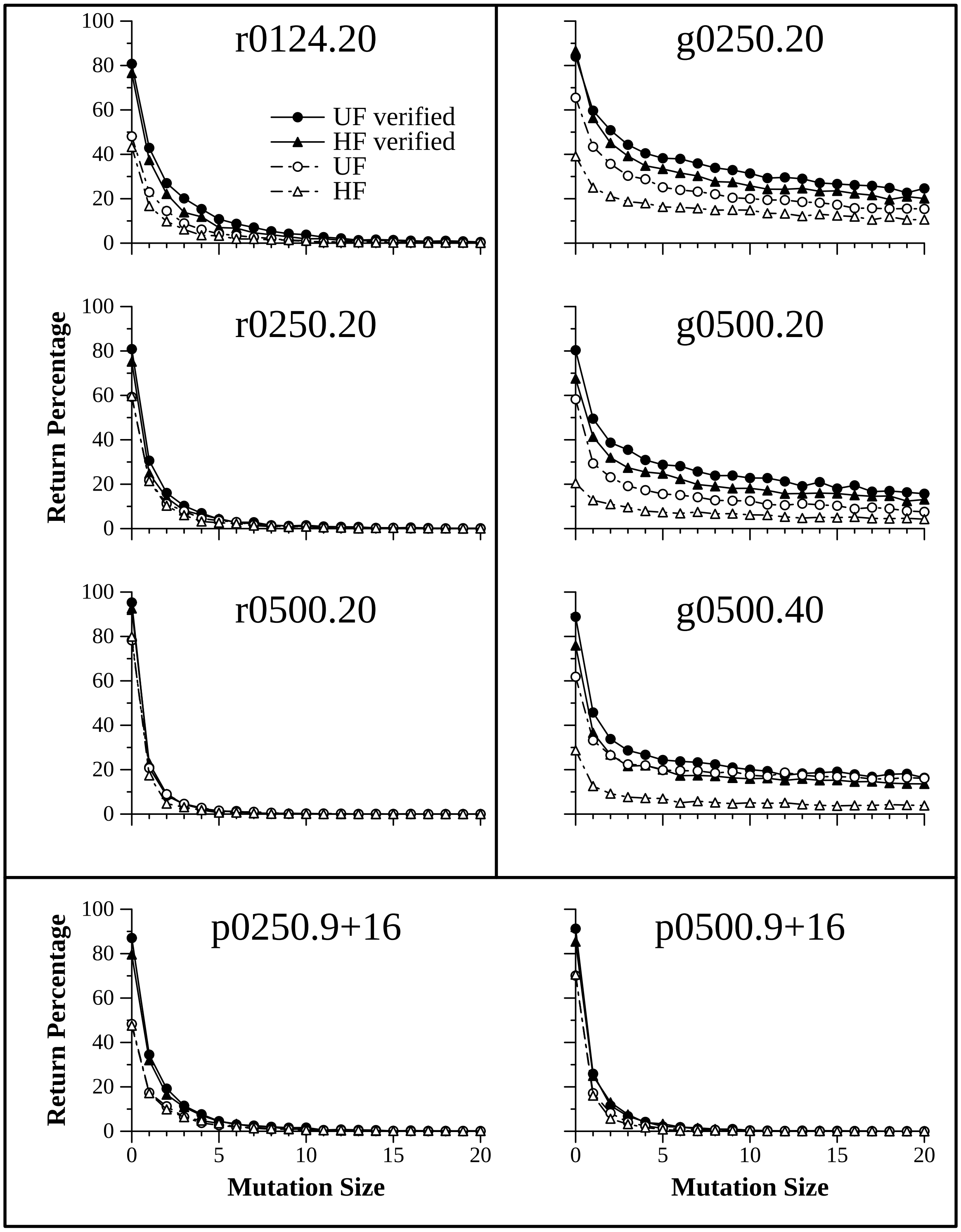

Figure IV.12: Return percentage for mutants

in determining proper mutation size.

A second reason the data is misleading has to do with the specifics of the local search methods. As described in Section IV.B.1, the **HF** and **UF** methods do not always result in a partition which is at a local minimum. In fact, after applying **HF** (or **UF**) to a partition, reapplying it to the result may produce a better solution. This is part of the explanation for the low percentages in Figure IV.12. Local search doesn't return the partitions to their starting points because these aren't necessarily local minima.

This claim is supported by Table IV.22, which shows the percentage of time the "local minima" from above are returned when local search is applied to themselves (the same experiment as above, but with mutation size zero). We see that for all graphs, there is a substantial chance of local search modifying the results of previous local search.

Table IV.22: **local minimum stability:** When local search is applied to partitions which are themselves the result of (one previous) local search, the percentage of time the same partition is returned. The columns labeled "one LS" are for partitions which are the result of a single local search. The "verified" columns are for partitions which have undergone several local searches, the "settling" columns displaying how many local searches on average. See the text for details.

	HF			UF		
	one LS	verified	settling	one LS	verified	settling
r0124.20	44.0	70.9	2.79	46.0	78.8	2.41
r0250.20	63.4	85.6	1.99	61.2	86.2	1.77
r0500.20	80.8	85.9	1.43	81.2	92.5	1.47
g0250.20	35.0	85.2	6.20	72.1	87.7	1.46
g0500.20	15.5	66.4	9.69	50.5	82.7	1.98
g0500.40	26.5	73.5	12.18	72.4	88.6	1.63
p0250.9.16	51.4	68.8	2.11	44.8	79.2	2.44
p0500.9.16	70.0	81.5	2.38	71.6	91.5	1.52

In order to account for this problem, another set of experiments is performed, similar to the above, but using "verified" local minima. As before, we generate 100 initial partitions and perform local search on each of them. Then we repeatedly perform local search on the results (updating the solutions if improvements are found) until five local searches in a row are performed without finding an improvement. In this way we obtain partitions which are partially verified to be true local minima. We will call these *verified local minima* and their corresponding basins *verified local basins.*

Table IV.22 shows the LS return percentages for the verified local minima next to the corresponding data for the unverified local minima. We see that in all cases the verified local minima are substantially more stable, in that local search is less likely to end up with a different partition. Note that the percentages are still less than 100% because of the equivalent partitions problem, and also because the verified partitions are not completely guaranteed to be local minima. Also displayed is the "settling time," or the average number of local searches performed before finding the final solution. This is seen to be quite high for the geometric graphs under HF, and this is reflected in the pronounced difference between the verified and unverified return percentages seen for the graphs.

The stability of the verified local minima under mutation is displayed in Figure IV.12 along with that of the unverified partitions. As expected, we see that in all cases the percentages are higher for the verified local minima. Despite the greater overall stability, however, the original observation about low percentages still applies. For most graphs, even a single node swap is usually enough to escape from a local basin. To some extent this is still an artifact of the "equivalent partitions," so it is difficult to infer a proper mutation size from the data.

As a final note, it is tempting to regard the verified local minima data as more useful, as it is more pure in terms of what it considers to be local minima. However, both sets of data (verified and unverified) are relevant to understanding the behavior of the EA+LS algorithm. The unverified local minima are what the EA will see in early stages, when population members have undergone only one LS, if that. After the population has largely converged, mutation and local search will be the driving forces, and most population members will more likely be at or near true local minima. This situation is better reflected by the verified local minima data.

Graph and LS characteristics Figure IV.12 and Table IV.22 contain useful comparative information regarding the different graph instances. First, note that the geometric graphs show a more gradual dropoff in stability than the other graph types do as the size of mutations increases. Consider the ratio of the return percentage after size one mutations vs. size five mutations. We see that this ratio is almost always less than 2.0 in the case of geometric graphs, whereas it ranges from 4.4 to 22.3 for random graphs and 6.2 to 13.8 for planted bisection graphs.

This slow dropoff for geometric graphs, together with the fact that the geometric graphs have higher return percentages overall, is a good indication that the local basins are more “well-defined” and perhaps larger in geometric graphs. By this we mean there is some local structure from which it is difficult to escape by small mutations. It may seem contradictory that the return probability is low (less than 40%) for five mutations, yet drops off slowly all the way out to 20 mutations. It seems to imply that there is not too much structure at a scale of more that 5 swaps, but the structure that *is* there extends to at least 20 swaps. Our interpretation of this is that there are indeed large basin-like structures (size 20 or more), but that within each of these there are the usual tiny basins containing roughly equivalent partitions. Mutations of various size traverse the tiny basins easily, but cannot escape the larger structure.

As further evidence for this view, consider Figure IV.13, which shows the average Hamming distance of the reoptimized mutants from their initial local minima. That is, after each mutant in the above experiments is locally optimized, its Hamming distance from the initial local minimum (before mutation) is measured. The figure shows a clear difference between the three graph types. For the geometric graphs, the reoptimized mutants are much closer to the initial local minimum than the mutants themselves. In other words, local search acts to return the mutants towards the initial optimum. Even after 20 mutations, local search finds optima which have an average Hamming distance of only 6.59 from the initial minimum. Note also that 20 mutations corresponds to a mutant Hamming distance of approximately 40, since each mutation swaps a pair of nodes (expected value slightly less than 40 since the same node may be moved twice).

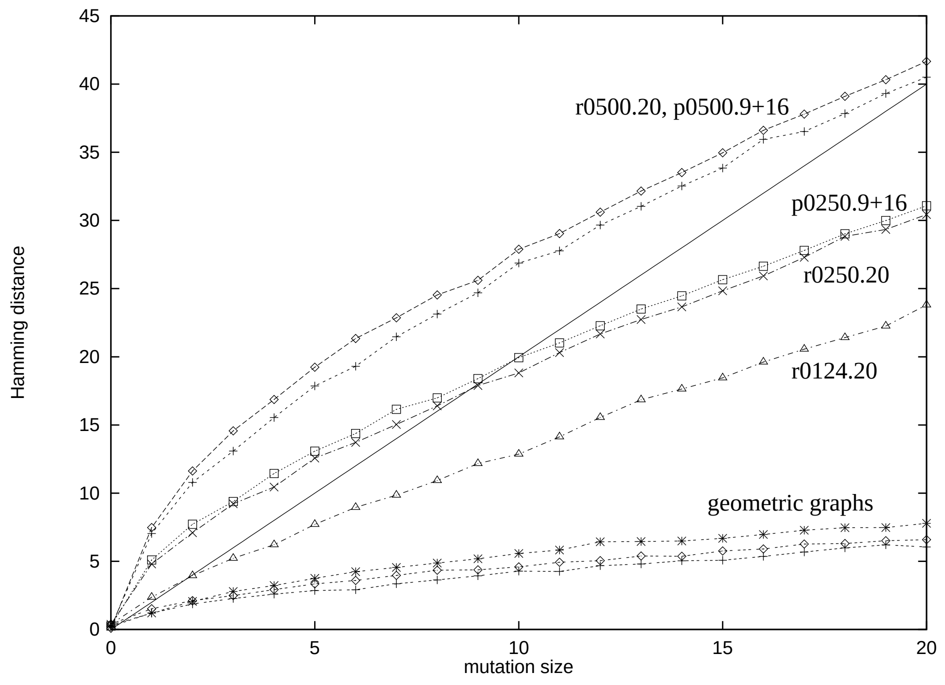

Figure IV.13: **Average Hamming distance of reoptimized mutants:** For each graph, the average Hamming distance of the reoptimized mutants from the initial local minimum, as a function of mutation size. The straight line approximates the expected Hamming distance of the mutants themselves, before local search.

For the random and planted bisection graphs, local search actually moves the mutants *away* from the initial local optimum in the case of small mutations. This is a clear indication that these mutations are outside of the initial minimum's basin of attraction. Note however that as the mutation size increases, local search does begin to move the mutants towards the initial minimum. The mutation threshold at which this happens depends on the graph size. We believe this is a manifestation of the fact that all local minima have a certain amount in common, creating a tendency for random points to be moved towards each other by local search. Note that while the size of these graphs affects the distance of the reoptimized mutants, for a given size there is little difference between the date for random or planted bisection graphs. Contrast this with the situation for the geometric graphs, in which the graph size makes little difference.

The data also allow a comparison of basin properties under different local search methods. The main observation here is that the partitions found by UF are generally more stable than those found by HF. This may be because UF does a more thorough search to begin with, and therefore has less room to improve.

In summary, it is difficult to get a clear idea from the data presented what mutation size may be appropriate. This is partially because of very small barriers between equivalent partitions, resulting in a proliferation of tiny basins which are mostly irrelevant to the global search. Another reason is the imperfect nature of the local search algorithms used, which do not always return true local minima. Despite these problems, some general observations can be made regarding the stability of local minima for various graphs and local search methods. Specifically, geometric graphs have more stable local minima than random or planted bisection graphs, and the UF method produces more stable minima than the HF method.

Effect of Mutation Size

In order to examine the role of mutation in the context of local search, experiments are performed using various mutation sizes in an EA+LS hybrid. The canonical mutation size is one swap (expected value) per genotype. We expect this to have little effect since the basins are presumably larger than this. A mutation size of zero is also tested for comparison. Due to the difficulty in determining basins sizes (see Section IV.D.3), somewhat arbitrary mutation sizes of five, ten, and twenty node swaps are chosen to test to effect "large" mutation. As has been seen, mutations of these sizes will usually disrupt local minima enough that local search will not return to the same partition.

Figure IV.14 compares EA+LS using the five mutation sizes on eight instances of graph partitioning with the HF local search method. Figure IV.15 displays data from corresponding EA+LS runs with the UF local search method. As can be seen, the largest mutation sizes generally result in the best final solutions. In all cases, these solutions are either statistically significantly better than or are statistically indistinguishable from the solutions found using smaller mutation sizes.

In contrast to the situation at the ends of the runs, we see that large mutations may cause the EA+LS to perform *worse* initially. In fact, for graphs in which there is a clear difference early on, using no mutation is best of all, and using large mutation is worst. We believe this is due to mutation partially undoing the effects of local search. Mutation serves to move the population away from local minima, which degrades the average population fitness. Note that since mutation is applied equally to all population members, this may not substantially hinder the global search in the early stages. If all members' fitnesses are affected the same amount, the relative comparisons between members will be unaffected, and the global selection should roughly focus on the same regions as it would otherwise.

The two observations in the preceding paragraphs confirm our predictions that larger-than-usual mutations are appropriate for an EA+LS hybrid. The benefit of such large mutations in the latter stages of a run are so substantial that they overcome the detrimental effect observed early on.

What is most surprising about these figures is that very large mutation sizes (20 node swaps per individual) are not detrimental. If the EA's population contains useful information about the search space, then disruptions of this magnitude might be expected to degrade the EA's effectiveness. How large can mutations be without being detrimental? In Section IV.D.4 we examine the extreme case of "macromutations", in which half the genotype (on average) is mutated.

IV.D.4 Effect of Crossover and Macromutation

In order to judge the role of crossover, we compare our standard EA to two variants. The first variant uses no crossover, so that a new solution is generated by simply cloning one parent and applying mutation. If crossover is of any benefit then this method will not do as well as the canonical EA. Note that removing crossover takes away one source of variation in the population. Hence, the EA will often

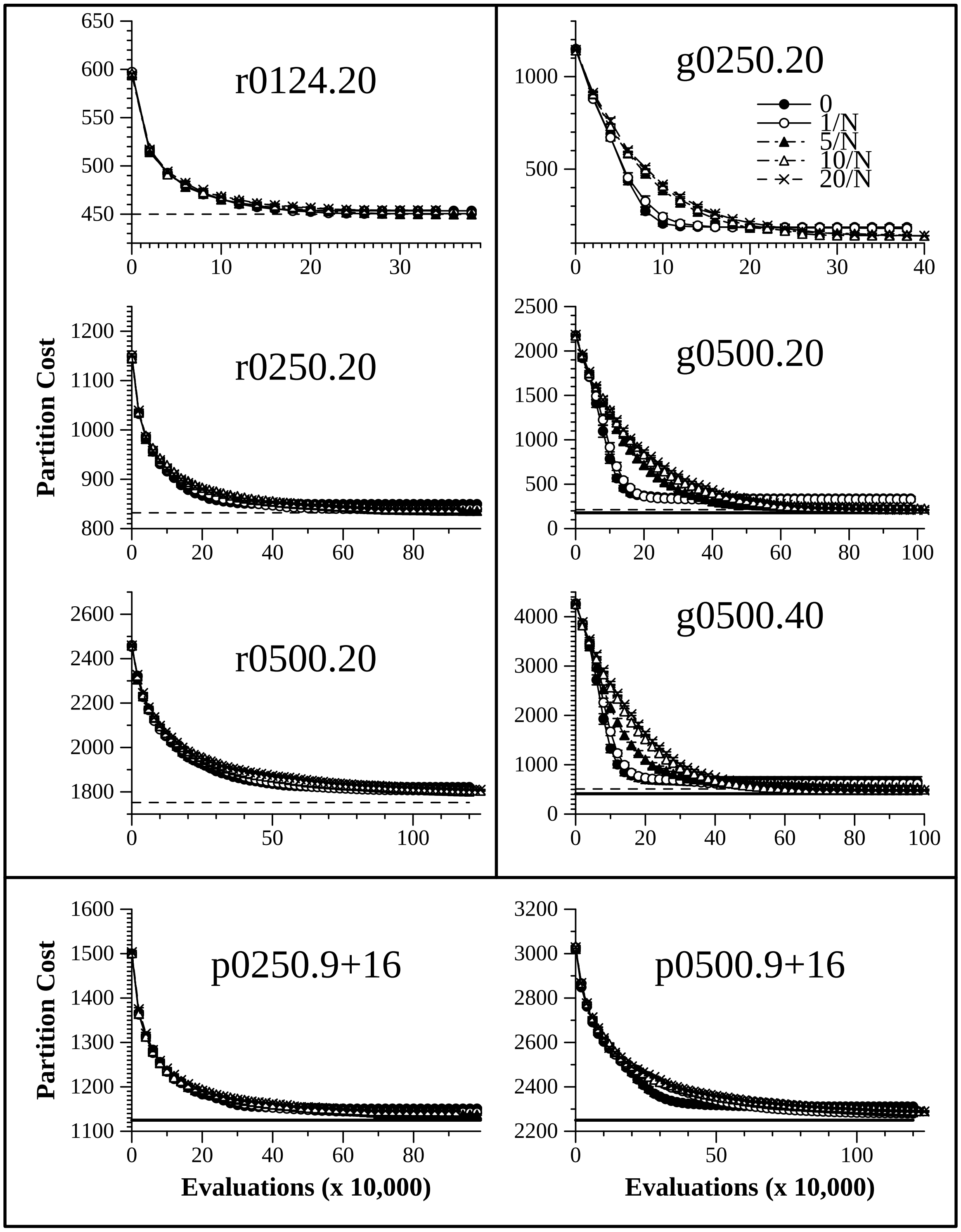

Figure IV.14: Mutation rate comparison using HF

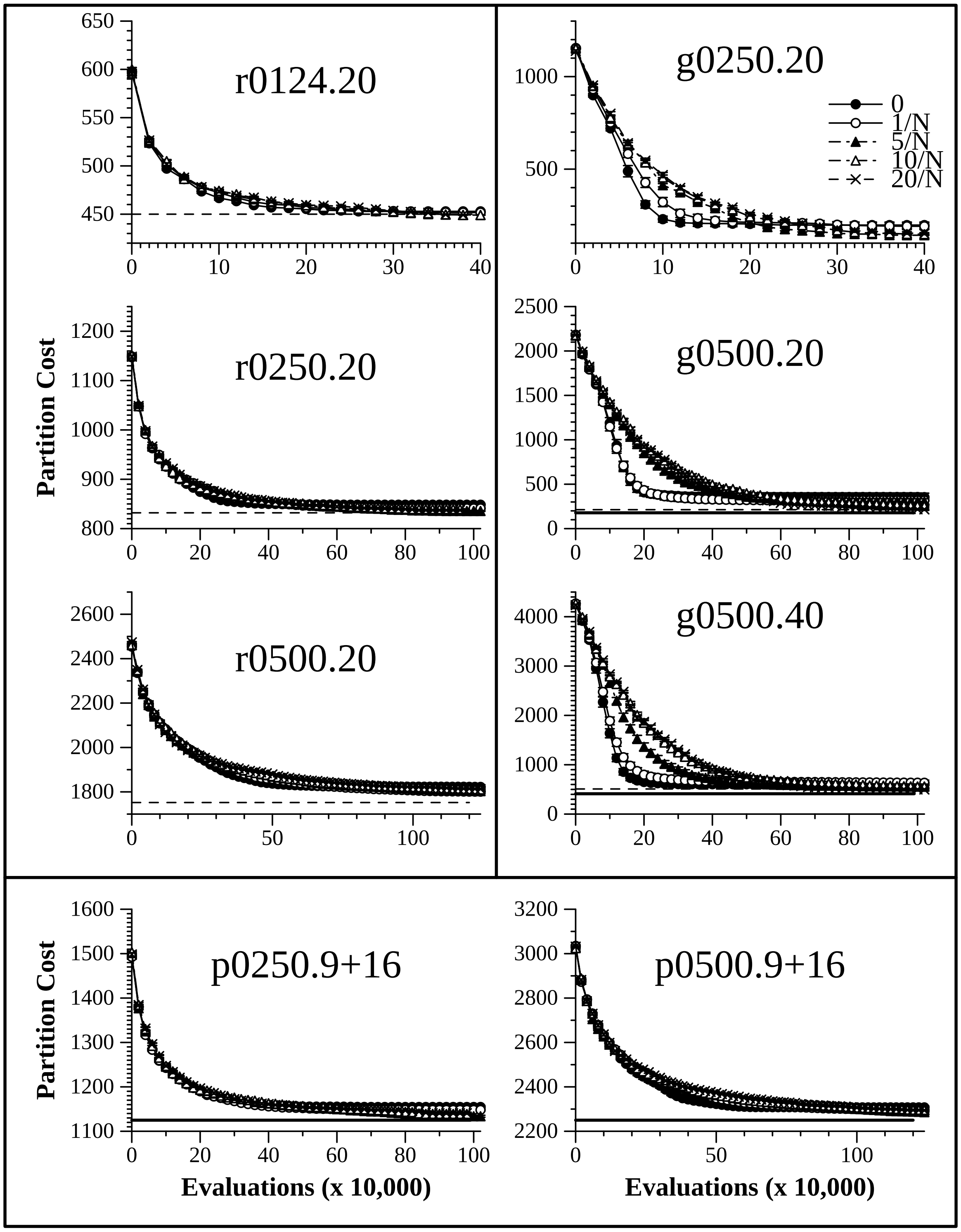

Figure IV.15: Mutation rate comparison using UF

converge quickly unless mutation is large enough to compensate.

The second variant we examine uses so-called "random" or "headless chicken" crossover [46], in which the standard crossover operator is used, but one of the parents is a new randomly generated solution. The purpose of this experiment is to determine *why* crossover is beneficial, if it is. Crossover typically is thought to combine useful building blocks from distinct solutions to produce a child with the best of both parents. If this is the case, then an EA using random crossover should not do as well, since it does not combine parts from different solutions in the population. On the other hand, crossover is sometimes beneficial simply because it allows large-scale variations, or "macromutations." If an EA using random crossover works as well as one using standard crossover, this is a good indication that crossover is not effective at combining building blocks.

Figure IV.16 compares an EA using standard-, random-, or no-crossover on eight graphs. The expected mutation size for these experiments is five node swaps per solution. This value was chosen as a good compromise between speed and solution quality based on the experiments in Section IV.D.3.

Surprisingly, we see that in general standard crossover does not do substantially better than either of the two variants. On the random and planted bisection graphs, standard-crossover and no-crossover perform approximately the same, with random the worst, though the differences are small in all cases. On the large geometric graphs, the order is reversed, with random doing the best, and no crossover the worst. For all graphs, no crossover finds the best solutions in the early stages of the runs.

To check that crossover's ineffectiveness is not simply a feature of our steady-state algorithm, we repeat the experiment using a generational EA+LS with simple elitism. We perform LS to completion on 5% of each population, chosen randomly. Because a generational EA has somewhat lower selection pressure, smaller mutation sizes are appropriate: we use one node swap (expected) per genotype. All other parameters are the same as in the steady-state case. The results are shown in Fig-

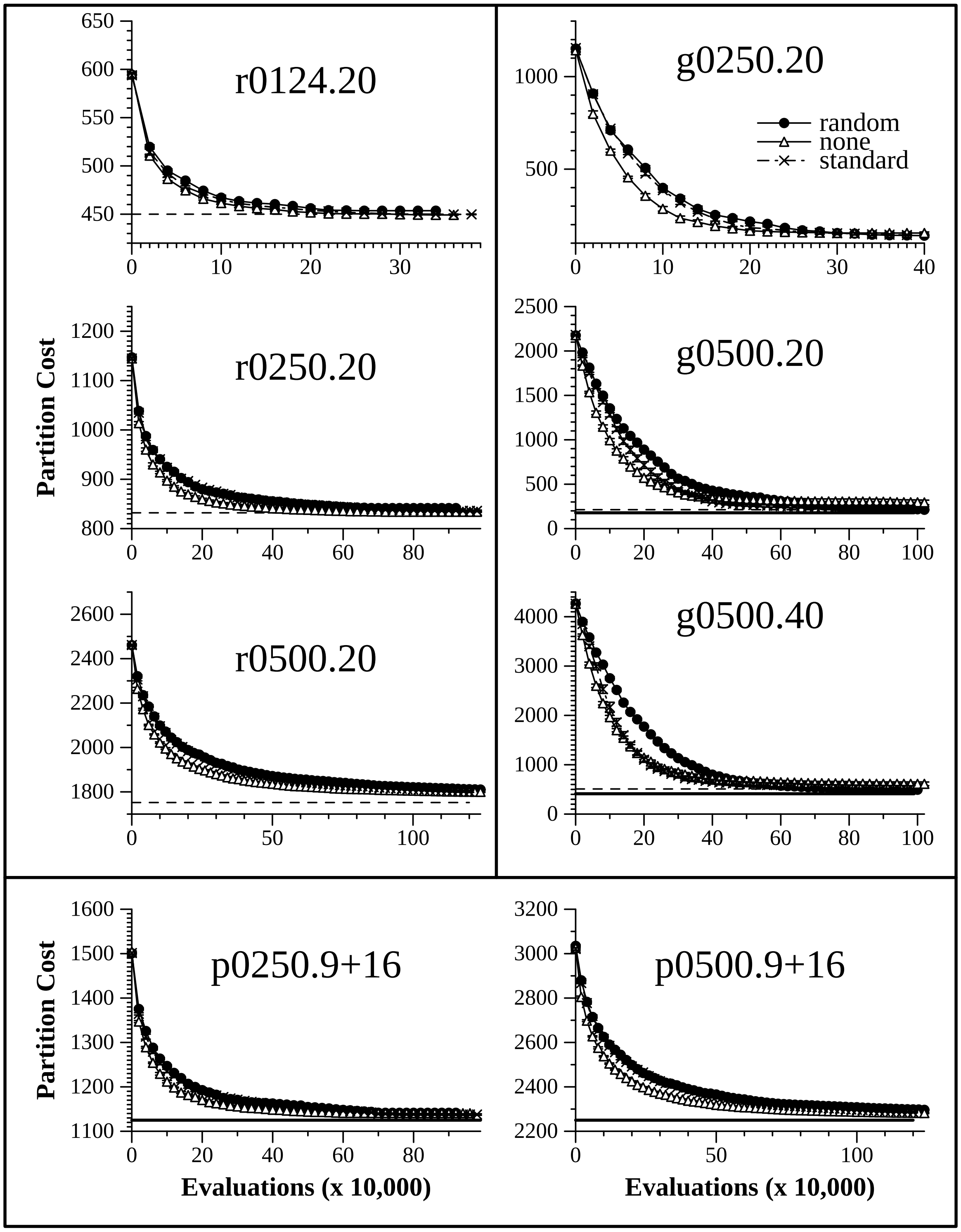

Figure IV.16: Comparison of crossover methods when using LS

ure IV.17. The same behavior is seen, there being little difference between standard-, random-, and no-crossover. Additionally, when the generational EA is run without LS, we do see (Figure IV.18) that standard-crossover is more effective than either random- or no-crossover, with random- being the worst by far. This mirrors the results of Section IV.C.2 using the steady-state EA+LS.

These results seem to imply that crossover is not operating as we expect it to, and is not even beneficial on some graphs. This is even more surprising in light of the fact that crossover is essential when we are not using local search (see Section IV.C.2). It seems that when local search is *not* used, there are *composable* building blocks which are combined by crossover, but that using local search causes this process to "break" somehow. We consider two possible explanations. The first is that the populations are too converged for crossover to work. The second is that local search itself exploits the compositional structure of the search space, obviating the need for crossover. We will show that the first hypothesis is incorrect for at least some of the instances, and that the second is consistent with all the data available.

Lack of Diversity? When local search is used in a Lamarckian fashion, one may expect the population to converge quickly. This can happen because local search often acts to reduce variation; there are commonalities across local minima, and therefore population members have more in common with each other after local search has been applied than before. Standard crossover has no effect on a fully converged population, as the offspring of two identical parents are unchanged. Furthermore, an EA's search is essentially over once the population converges, unless the genetic operators create enough variation. Hence, random crossover may be quite beneficial in this situation.

Figure IV.19 displays a measure of the population diversity for runs using each of the three crossover methods. The diversity measure used is the average normalized Hamming distance (as defined in Section IV.A). We see from the figure that there is a strict ordering of the three methods in terms of diversity: in all cases random-crossover maintains high diversity throughout, no-crossover quickly loses its

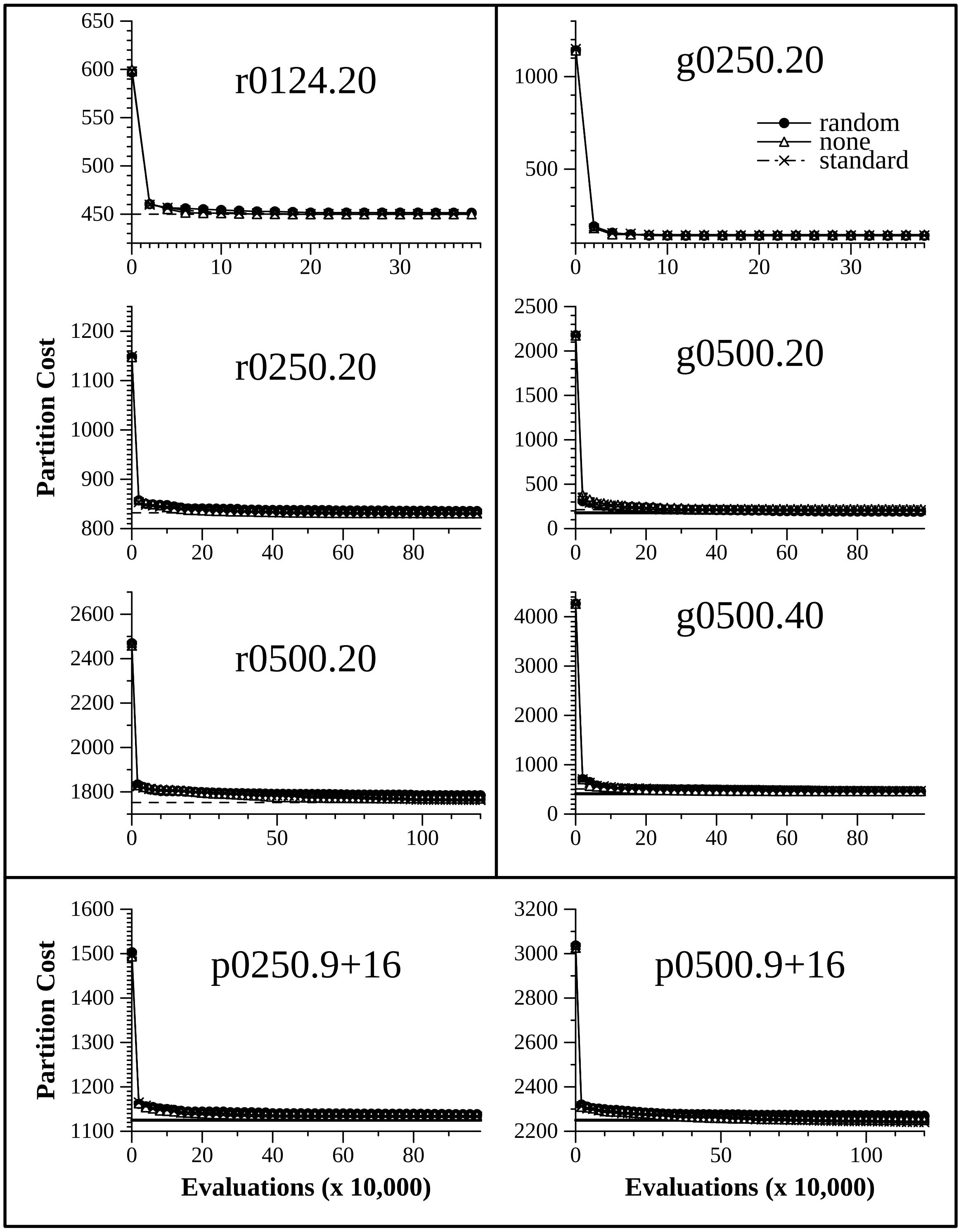

Figure IV.17: Comparison of crossover methods for a generational EA using LS

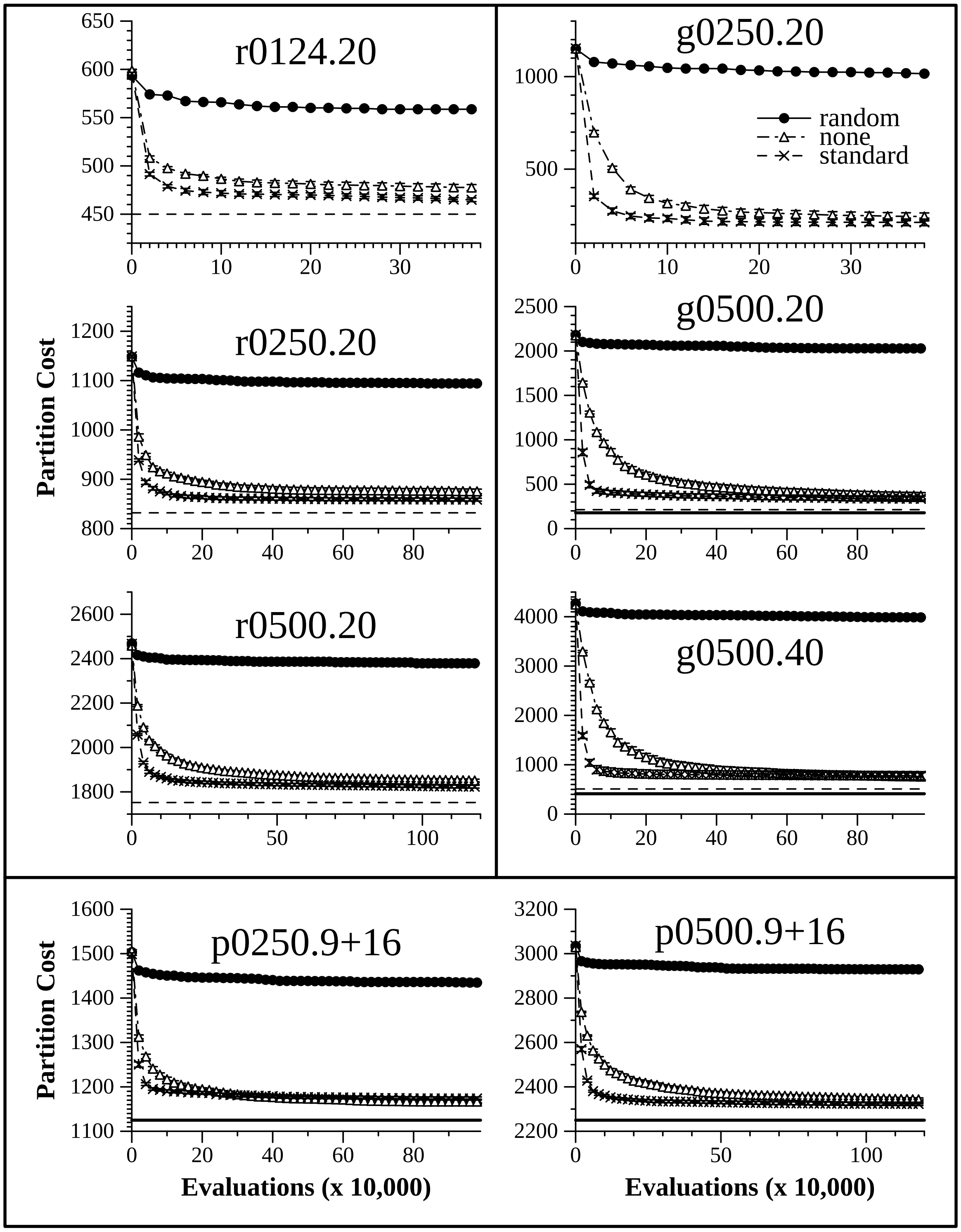

Figure IV.18: Comparison of crossover methods for a generational EA without LS

diversity, and standard-crossover falls somewhere in between. This is as it must be, reflecting the relative amounts of variation introduced by the three methods.

A more revealing observation from Figure IV.19 is that the speed of convergence under standard crossover depends on the graph. Specifically, for the random and planted bisection graphs the standard EA retains high diversity late into the runs. This runs counter to the proposed explanation for crossover's ineffectiveness, namely that there is not enough variation across the population for it to combine disparate solutions in novel ways. Note that on the geometric graphs diversity drops off quite quickly with standard crossover, almost as quickly as with no crossover. Despite the fact that this is consistent with the proposed explanation, we believe that there is more going on. Our second explanation, discussed next, is consistent with the convergence rates for all types of graphs.

LS Exploits Compositional Structure? The second explanation concerns the nature of the local minima in typical graph partitioning instances. The local minima contain many similarities to each other: there are certain schemata (e.g. two particular nodes together or apart) which are common to most minima. As a contrived example, consider a graph with a small clique, which is not connected to the rest of the graph. Local search from any initial partition will place all nodes of this clique together if it can do so while maintaining balance. Hence all local minima will have this pattern (or schema) in common. Furthermore, if there are several such cliques local search will place the nodes of each one together.

If an EA without local search is applied to such a graph, these schemata will have to be discovered through the standard processes of random variation and selection. The key point is that crossover allows these separate building blocks (cliques) to be discovered independently and then combined. Crossover is hence crucial to the global search. When local search is used, however, all of these building blocks are found automatically by each complete local search. If there are no larger scale building blocks (too large for local search to reliably handle) to be combined, then

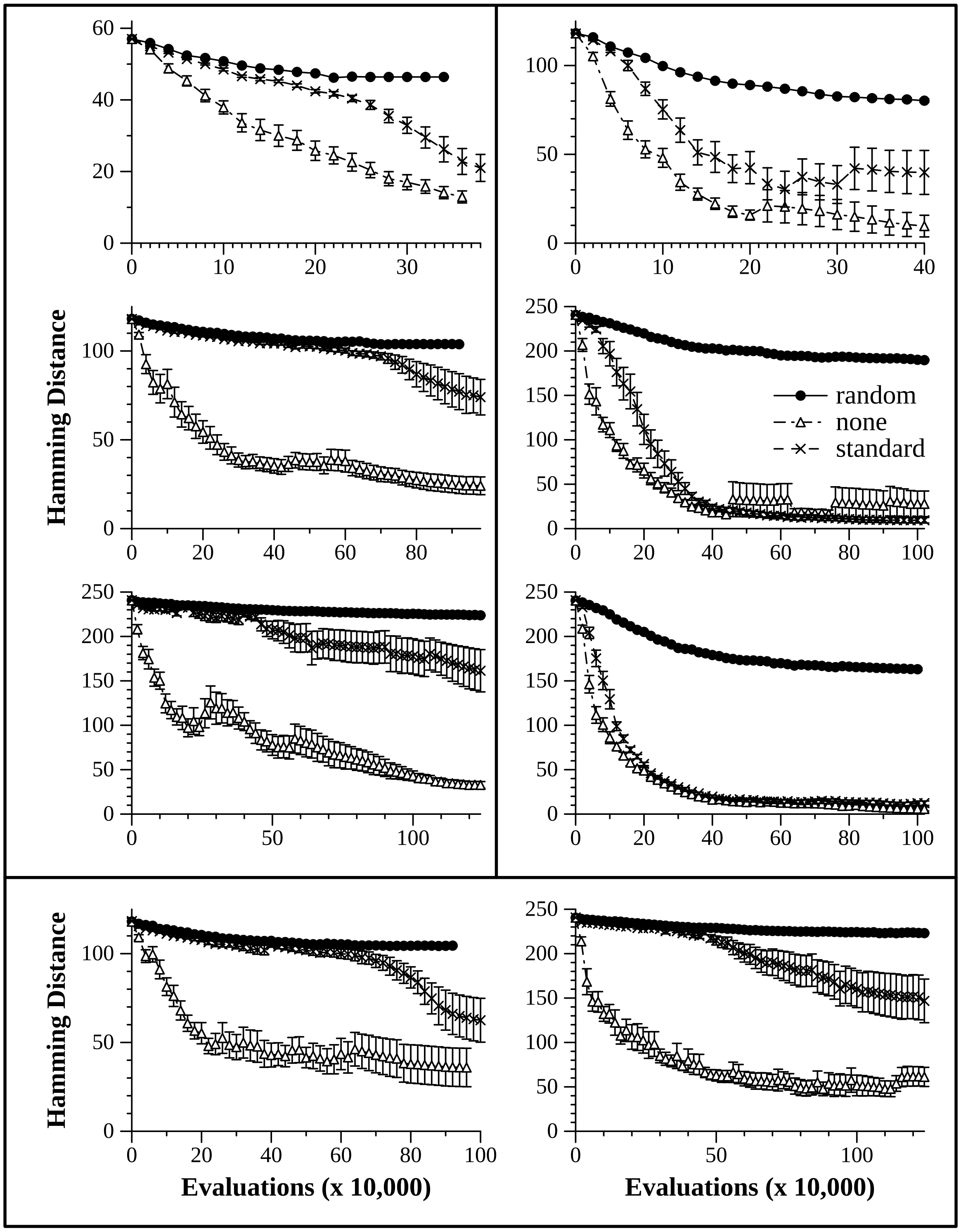

Figure IV.19: Population diversity for various crossover methods

crossover will not be useful for this purpose. This is of course an extreme example but it provides a sense of how a single use of local search can discover multiple good building blocks. Note that crossover *may* still be useful for generating variation, however, and in this case random crossover may do as well or better.

This hypothesis is somewhat astonishing, and counters our original expectations based on our intuitions and especially the results of Section IV.C.2. Yet it is consistent with all the data presented in this section and Section IV.D.3 regarding the role of mutation. As we saw there, large mutation sizes (five node swaps per solution) are beneficial, and very large mutation sizes (ten or twenty node swaps per solution) produce quite similar results, in some cases a little better. In fact, the largest mutation size examined produces results which are comparable to those of random crossover. Roughly, as long as mutation gets out of the local basin, it makes no difference how big a change it makes. This seems to imply that there is little exploitable structure to the search space, beyond that handled by local search. The possible generality of the hypothesis to other problems is discussed in Section V.

IV.D.5 Local/Global Search Ratio

The EA+LS hybrid is a combination of global and local search search methods. As such, a question immediately presents itself: what it the appropriate amount of effort to put forth doing local as opposed to global search? We explore this issue by varying the *local/global ratio*, which is defined to be the number of evaluations spent doing local search each time a new solution is generated by the genetic operators.

All experiments described so far in this section have used the same procedure for allocating local search; each time a new solution is generated by the genetic operators, ten steps of LS are applied to it, and then two randomly chosen solutions in the population also get ten evaluations each. The parameters used (ten evaluations, two extra individuals) are somewhat arbitrary.[3] We will investigate the behavior of

[3]In fact, the decision to perform LS on two individuals was made so as to maintain the same local/global ratio as was observed in previous experiments employing a generational EA with infrequent, complete LS.

the EA for various values of this ratio, as well as the LS increment.

The local/global ratio is determined by three algorithmic parameters: the amount of LS given a new solution when it is created (the *sniff length*), the number of additional solutions chosen to get LS (the *LS rate*), and the amount of LS given to each of these additional solutions (the *LS increment*). Simply,

$$\textit{local/global ratio} = \textit{sniff lenth} + (\textit{LS rate})(\textit{LS increment}).$$

Hence, the local/global ratio for all experiments described so far has been 30. In order to get a sense for how appropriate this ratio is, we compare with runs using local/global ratios of 10 and 100. Regarding LS increment, the default value of ten intuitively seems like a lower bound on how much LS is useful—longer searches may be beneficial. Therefore, for each local/global ratio, we try LS increments of 10 and 100. Finally, since the solutions getting LS are selected randomly, the initial "sniff" may not be beneficial. We perform experiments with sniff length 0 and 10. In Section IV.D.6 we examine other selection methods for which the sniff is expected to be more important.

Table IV.23 summarizes the experiments performed. Note that some combinations of parameter values result in a fractional LS rate. The EA handles this deterministically by keeping a running floating point sum which is incremented by the LS rate whenever a new solution is generated. Each time the sum surpasses an integral value a solution is chosen for local search.

Figure IV.20 compares the six methods which have sniff length zero. Early behavior is determined primarily by the LS length while the final solution quality (when there is a difference between the methods) is determined mainly the local/global ratio. More specifically, for all graphs an LS increment of 100 gives significantly better solutions early on than LS increment 10. For the random and planted bisection graphs, all methods are approximately equal in terms of the final solution. On the geometric graphs, however, the larger local/global ratios result in better solutions. The effect of LS increment, which is so strong early in the runs, is minimal by the end.

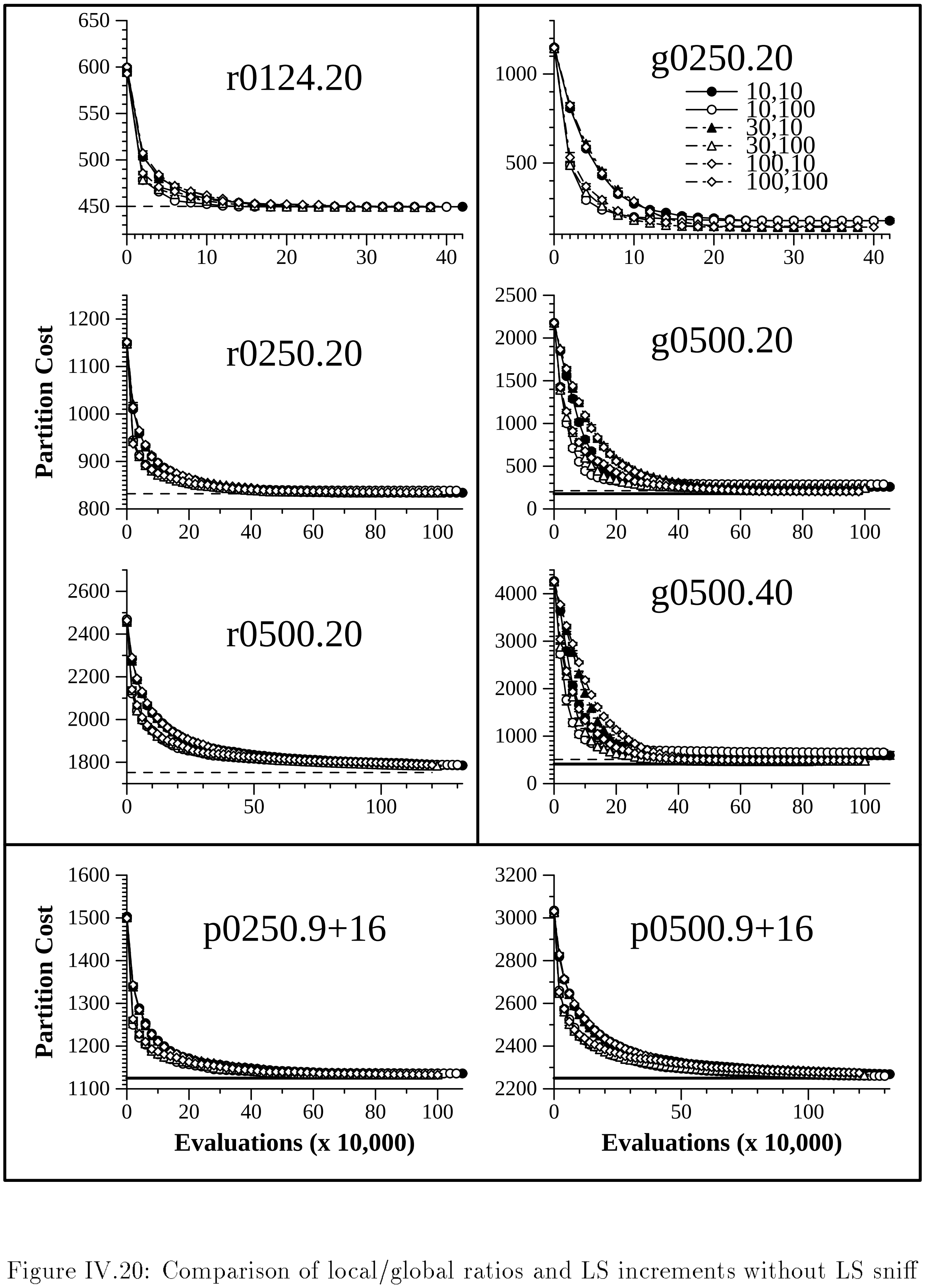

Figure IV.20: Comparison of local/global ratios and LS increments without LS sniff

Table IV.23: **Experiment summary:** The algorithmic parameter values used in the experiments of Section IV.D.5. For each experiment, the sniff length, local/global ratio, LS increment, and LS rate are given. Note that the experiment with sniff length 10 and local/global ratio 10 does no additional LS, so the LS increment is not applicable.

sniff length	local/global ratio	LS increment	LS rate
0	10	10	1
0	10	100	0.1
0	30	10	3
0	30	100	0.3
0	100	10	10
0	100	100	1
10	10	NA	0
10	30	10	2
10	30	100	0.2
10	100	10	9
10	100	100	0.9

Our interpretation of the first observation (longer LS helps initially) is as follows. From previous experiments we know that LS is quite powerful at quickly finding high-quality solutions (cf. Section IV.B.2). Hence, doing a long LS (100 evaluations) on a single random solution will generally yield a better partition than doing short LS (ten evaluations) on ten random solutions. This is manifested in the EA, where focusing the LS effort on a small number of partitions is the fastest way to improve initially.

In the longer term the benefit of long LS does not persist simply because there is a limit to the amount of LS which can be performed on a given solution. Even if LS is applied for only a few evaluations at a time, multiple applications will eventually result in reaching a local minimum. Furthermore, long LS may cause the population be become "unbalanced," with some members much better than others simply by virtue of having had more local search. This may skew selection and therefore harm the global search. In fact this effect is not seen in the data; for most graphs and local/global ratios, there is no statistical difference between using long and short LS in terms of the final solutions.

What does sometimes differentiate the methods at the end of the runs for geometric graphs is the local/global ratio. The differences are small, but in all cases a ratio of 10 results in worse solutions than a ratio of 30 or 100. On g0250.20 and g0500.20 the difference between a ratio of 10 and 100 is statistically significant. No such effect is observed on the random or planted bisection graphs.

Figure IV.21 displays the the results of the five methods which use a sniff length of ten. The first observation is that the worst method in all cases is that with ratio 10. This is not surprising, as no additional LS is performed other than the initial sniff. No solution ever gets more than ten evaluations worth of local search, unlike the other methods in which the local search effort applied to a solution can accumulate over multiple applications. Hence, it is difficult for local minima to be found with this method. Otherwise, the same observations apply as for the case with sniff length zero. Namely, LS increment 100 is always superior to LS increment 10 in the early stages, and larger local/global ratios are superior to smaller ones when there is a significant difference.

In order to examine the effect of the sniff length, we compare a sniff length 0 method with a sniff length 10 method. In both cases we use local/global ratio 100 and LS increment 100—these values are chosen because they are competitive with other methods on all graphs, for either value of sniff length. Looking at the final solution values, sniff length has virtually no effect, there being a significant difference for only one graph.

In summary, LS increment has a large effect on the speed with which the EA finds good solutions, with longer searches finding good solutions faster. This effect does not persist to the end of the run, however. Rather, the most important factor in final solution quality is the local/global ratio, with larger ratios being as good as or better than smaller ratios. This effect appears only for the geometric graphs, and is fairly weak even then. Finally, for an effective choice of local/global ratio and LS increment, there is no significant benefit or detriment to using an initial LS sniff.

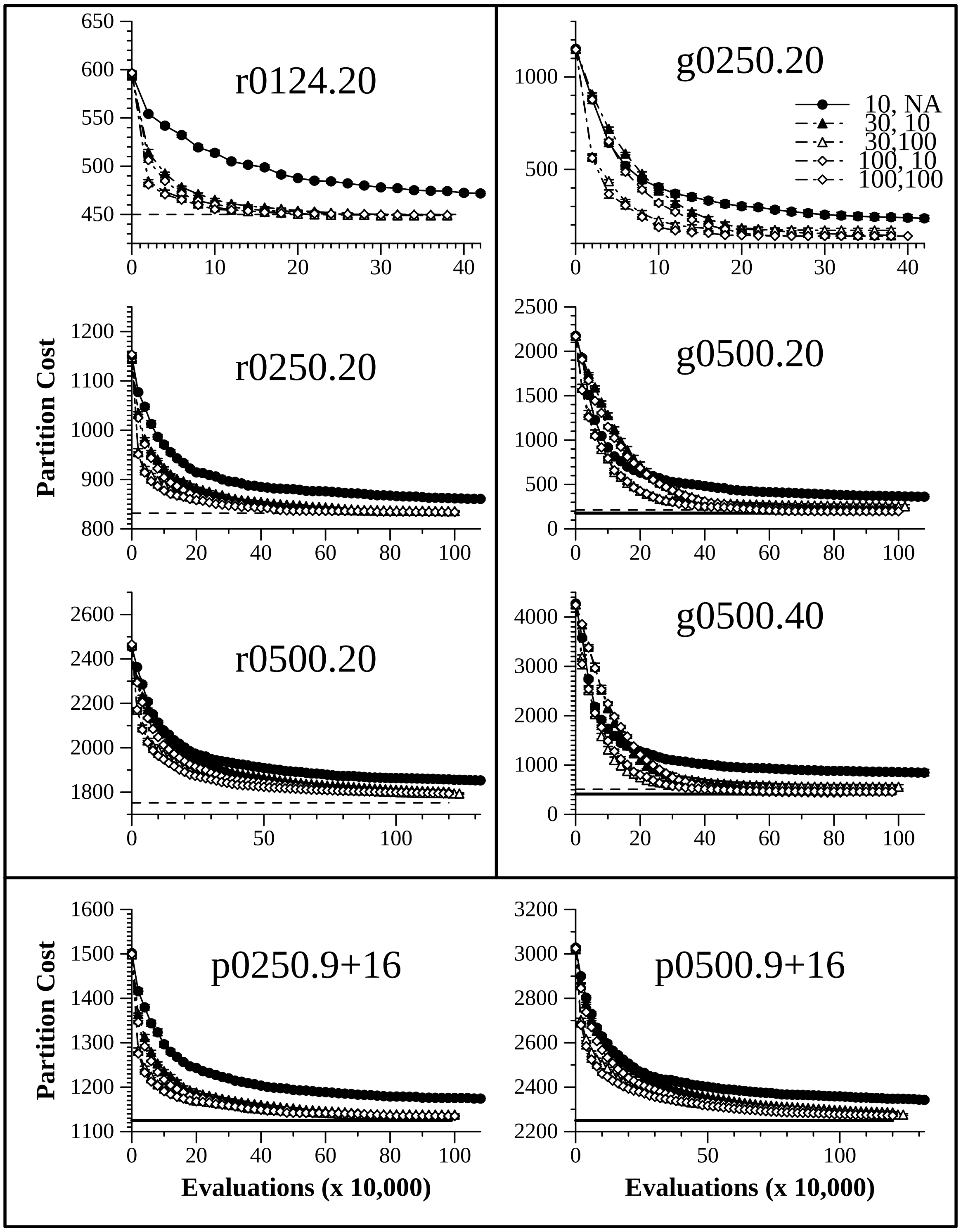

Figure IV.21: Comparison of local/global ratios and LS increments *with* LS sniff

IV.D.6 Choice of Local Search Selection Method

In many EA+LS algorithms, including ours, there is a step in which members of the population are selected to undergo local search. In typical generational implementations (cf. [38, 71, 56]), for example, some small fraction of the population is chosen at the start of each generation. The steady-state complete LS algorithm described in Section IV.D.1 does local search with small probability on each new solution as it is generated, while the steady-state short LS algorithm we use in most experiments picks k population members each time a new solution is generated.

All experiments described to this point have chosen the solutions which undergo LS uniformly at random from the population, without regard for fitness or previous LS results.[4] Section III.C.3 discusses several alternative strategies, including some previously examined by Hart [38] for continuous optimization. We refer to such strategies as *local search selection methods*. In this section we empirically evaluate two such methods: biasing LS selection towards fit solutions, and biasing towards solutions which have the greatest "local search potential." These methods are compared against our default method, random LS selection. Additionally, since both fitness and measures of LS potential are affected by the amount of LS a solution has received, various LS sniff lengths are used for each method. A partial examination of sniff length under the random LS selection method was explored in Section IV.D.5.

Preliminaries Define local search potential to be the amount of improvement which can be made to a solution by local search, before reaching a local minimum. There is of course no way to know this value without carrying out the local search, so we need some way to estimate it.

Related work by Carson and Impagliazzo [14] has shown that for some "hard" instances of planted bisection, a solution's LS potential is negatively correlated with its distance from the planted bisection. This provides some justification for focussing search on the solutions with greatest potential, as these are likely near-

[4]The exception to this is that we never select solutions on which LS has previously run to completion.

est to the global optimum. This observation does not provide us with a measure of potential, however, unless we happen to know what the planted bisection is ahead of time.

The method we use is to keep track of the success of any previous LS on the solution, and to equate *recent rate of improvement* with the LS potential. The motivation for this estimate is that LS typically makes rapid improvements when the solution is far from its corresponding local optimum, as there are many ways to improve such a solution; in the extreme, a totally random solution can be expected to have half its neighbors better than itself. As LS gets closer to the optimum, fewer and fewer improving moves are possible, and the rate of improvement decreases. Note that this applies to both first-improve LS, in which the number of neighbors which must be examined increases, and steepest descent LS, in which the best move is likely to have a smaller gain in fitness than when half of all moves are improving.

The specifics of our estimate are as follows: a running window of the 100 most recent evaluations of LS is kept for each solution in the population. The average decrease in partition cost per LS evaluation, taken over the most recent 100 evaluations, is used as our estimate of LS potential. For solutions which have had fewer than 100 evaluations of LS, the average is taken over their entire LS history. New solutions which have had no LS are assigned the average of the LS potentials of all population members which have had LS. Finally, solutions for which LS has run to completion are excluded from LS selection, and their LS potential is not included in the average assigned to new solutions.

When either fitness or LS potential is used for LS selection, the procedure used for this selection is the same as that used for reproductive selection (recall Section IV.D). In the case of LS potential, the "fitnesses" used for LS selection are simply the negative of the LS potential; we use the negative because the selection procedure is designed for minimization, and we want to bias selection towards solutions with the highest potential.

Finally, the experiments in this section use LS sniff lengths of 0, 10, and

20. In all cases, the local/global ratio is 30, and the LS increment is 10. Recall from Section IV.D.5 that

$$\textit{local/global ratio} = \textit{sniff lenth} + (\textit{LS rate})(\textit{LS increment}).$$

Therefore, the number of additional solutions chosen for LS each time a new solution is generated (the LS rate) will vary: the rate will be 3, 2, or 1 for LS sniff lengths of 0, 10, or 20, respectively.

Effect of Sniff Length

Random LS selection Figure IV.22 compares three sniff lengths for an EA+LS using random LS selection. On all graphs, the best results are obtained by using no sniff at all, and the worst results occur with the longest sniff length. It is not obvious that the sniff length should have any effect at all, as the remaining LS is simply being distributed randomly. Note, however, that longer sniff lengths have the effect of distributing LS more uniformly among the population, as every member gets at least the sniff. With no sniff, some members may never get any LS, while others will get much more than average just by chance. This seems to imply that a uniform allocation of LS is suboptimal, at least when the alternative is to allocate LS randomly without bias.

In order to get a better idea of how the search progresses, we examine the population at regular intervals and record how much LS has been applied to each solution (we call this the solution's *LS allocation*. This data tells us how uniformly LS is distributed among the population. We expect new members of the population to have undergone only a small amount of LS, while solutions which have been in the population longest will have accumulated the most LS just by chance. Figure IV.23 displays the average and standard deviation of the LS allocations for a single run on the graph u0500.20, using sniff length 10. The average and standard deviation are taken over the population, but do not include those solutions which have had only the initial LS sniff, nor those for which LS has run to completion. The number of

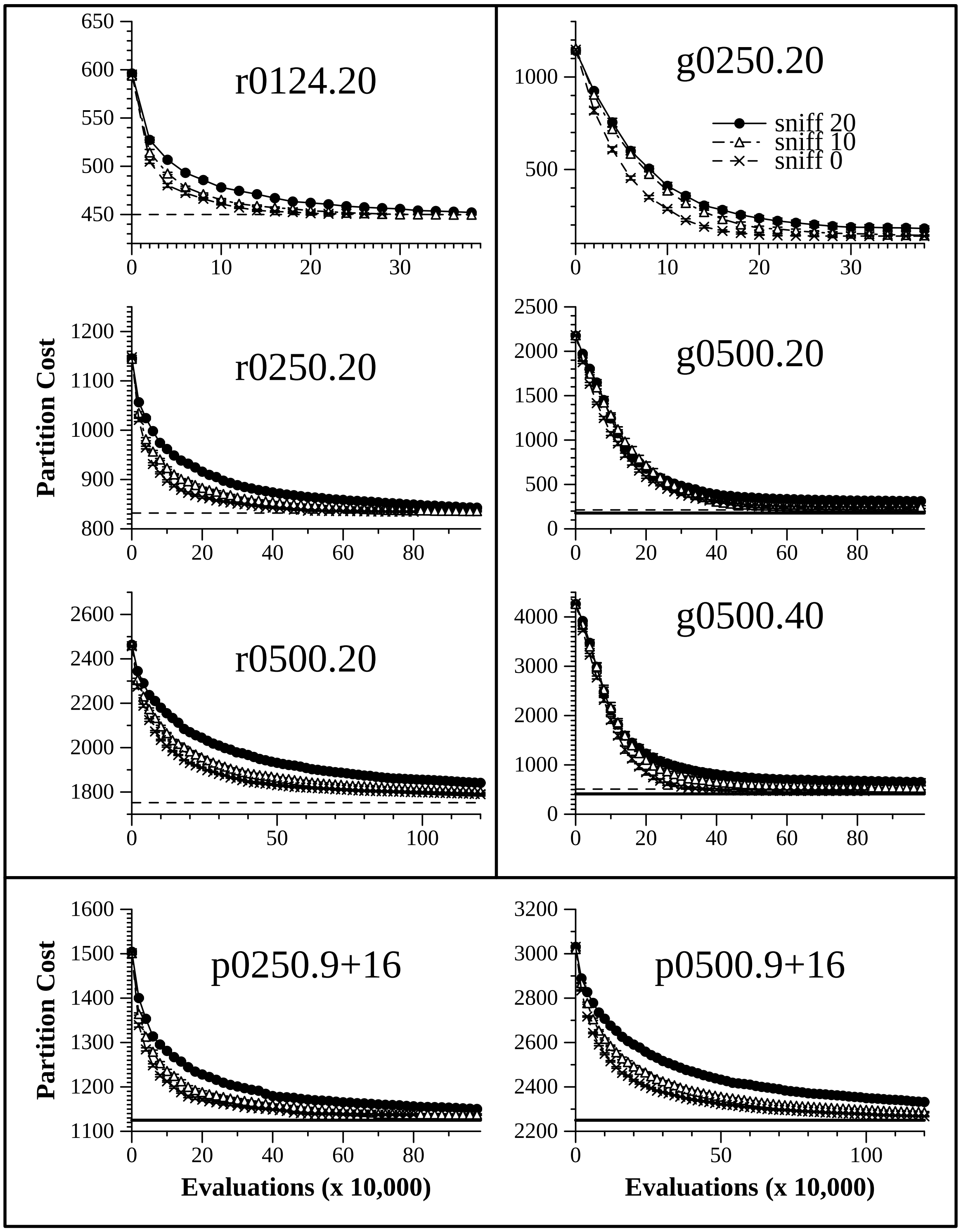

Figure IV.22: Sniff length comparison using random LS selection

solutions falling into each of these two categories is displayed with separate lines in the figure.

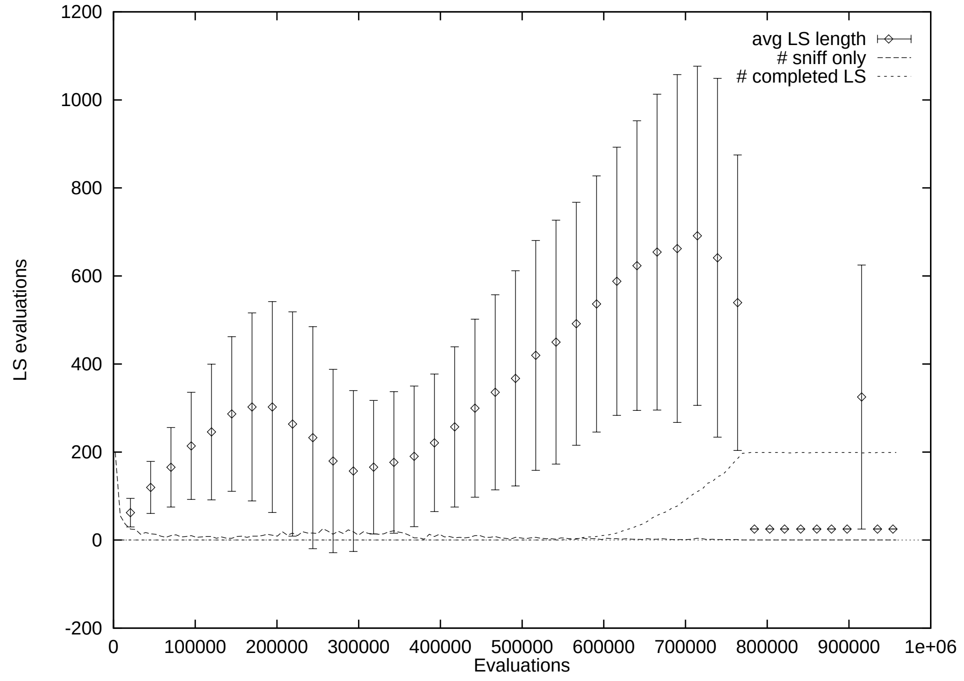

Figure IV.23: Distribution of LS allocation for random LS selection

Initially, we see a large number of solutions which have had only the LS sniff. The average LS allocation increases quickly, however, and soon most of the population has received some additional local search. After roughly 200,000 evaluations the average LS allocation drops sharply, and then begins rising again, and continues to increase throughout the run. By about 600,000 evaluations, there are several solutions for which LS has run to completion. The number of these solutions increases steadily until the entire population (other than the new solution each time) is at local minima. Note that this causes the average LS allocation curve to drop, since the local minima are not included in its average.

The general trend of increasing LS allocation is not surprising; as the search progresses, some solutions remain in the population for long periods by virtue of high fitness. These will periodically be allocated LS, thus increasing the average LS allocation. The dip in LS allocation, however, is unexpected. It implies that solutions which have been in the population a while and which have received substantial local search are suddenly being displaced by newer solutions. The phenomenon is also observed in other runs on the same graph, though at different points in different runs. Why this happens is unknown, but comprehending it seems important to understanding the progression of the search. This is a possible direction for future research.

Biased LS selection The next set of experiments uses fitness-based LS selection, again with sniff lengths of 0, 10, or 20. For these runs, the same selection procedure is used for LS selection as is used for reproductive selection (see Section IV.D). Therefore, the solutions which contribute the most genetically to future generations also get the most LS. As discussed in Section III.C.3, we may therefore expect faster convergence of the population and perhaps worse final solutions.

Figure IV.24 compares fitness-based LS selection for the three sniff lengths. Once again we see that larger sniff lengths result in worse solutions, when there is a difference. It is not clear if this implies that fitness-based LS selection is beneficial (as shorter sniff lengths allow more LS to be assigned based on fitness), or if we are simply observing the benefit of less uniformly allocated LS, as in the random LS selection case. Note, however, that the solutions which have received the most LS are likely to be the most fit, and therefore are the most likely to receive future LS. In this way the allocation of LS may be even more nonuniform than in the random selection case.

To examine this hypothesis, we display the LS allocation average and standard deviation in Figure IV.25 for a typical run on u0500.20, using fitness-based LS selection and sniff length 10. Two main differences are observed between this figure and Figure IV.23, which shows the corresponding data for random LS selection. First, both the average LS allocation and the standard deviation are much higher for the fitness-based run. This is consistent with the hypothesis that fitness-based LS selection leads to a more skewed distribution of LS allocation. Second, there is no dip in the LS allocation curve in fitness-based case. This may be related to the

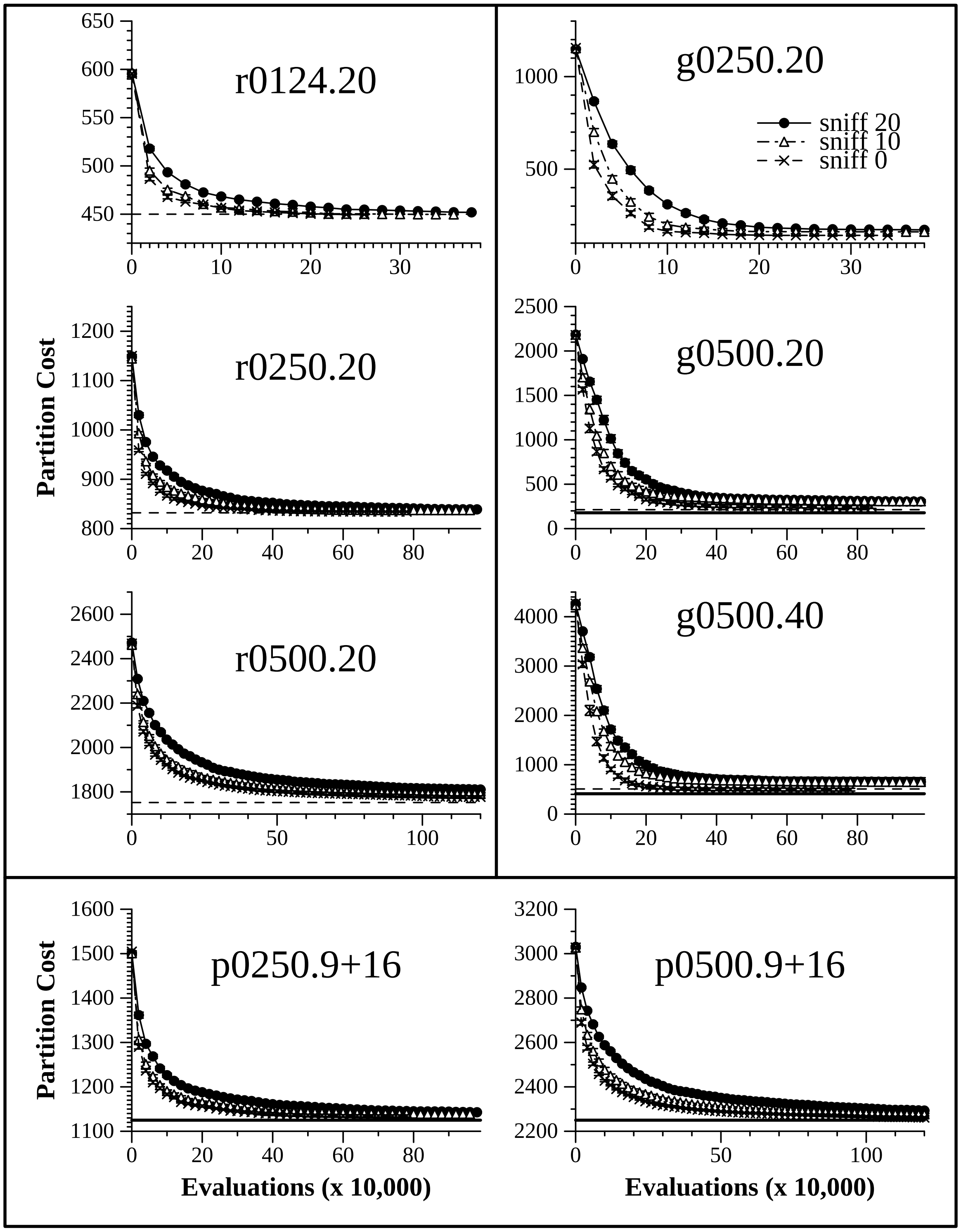

Figure IV.24: Sniff length comparison using fitness-based LS selection

possibility that long-lived solutions receive more LS than in the random LS selection case; since the established solutions have the benefit of extra LS, it is more difficult for new solutions to make to compete with and displace them. This explanation is speculative, however, as we do not yet understand the cause of the dip in the random case.

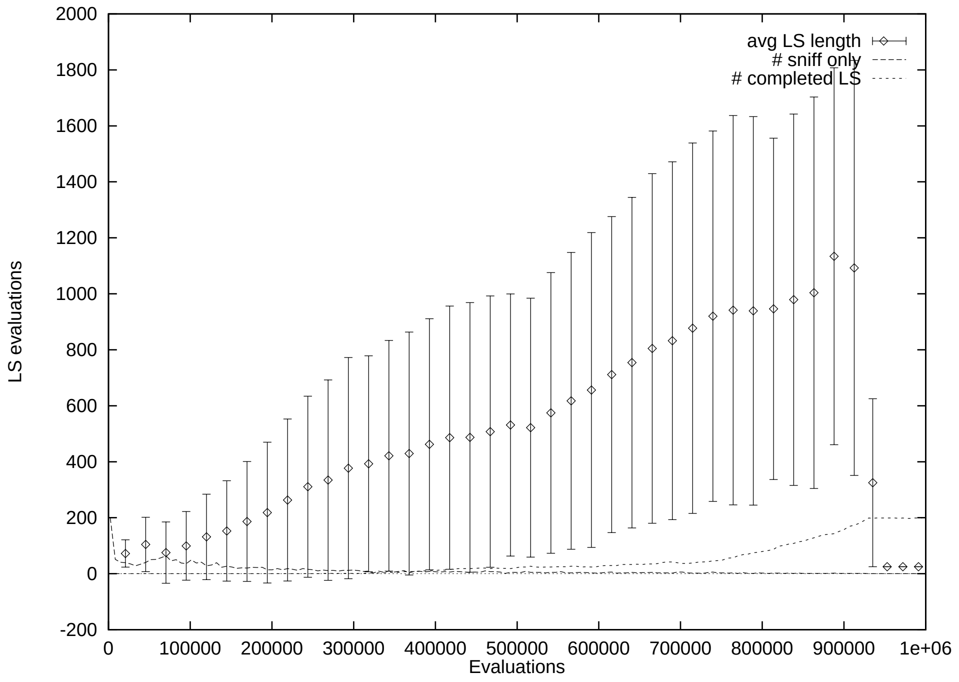

Figure IV.25: Distribution of LS allocation for fitness-based LS selection

Returning to the issue of sniff length under biased LS selection, Figure IV.26 displays the performance of the LS potential method under various sniff lengths. As with the other methods, when there is a difference in performance, longer LS sniffs lead to worse solutions. This relation is especially surprising for the LS potential method, as presumably some amount of LS is necessary to get an estimate of a solution's future potential for improvement. We will discuss possible explanations for this below.

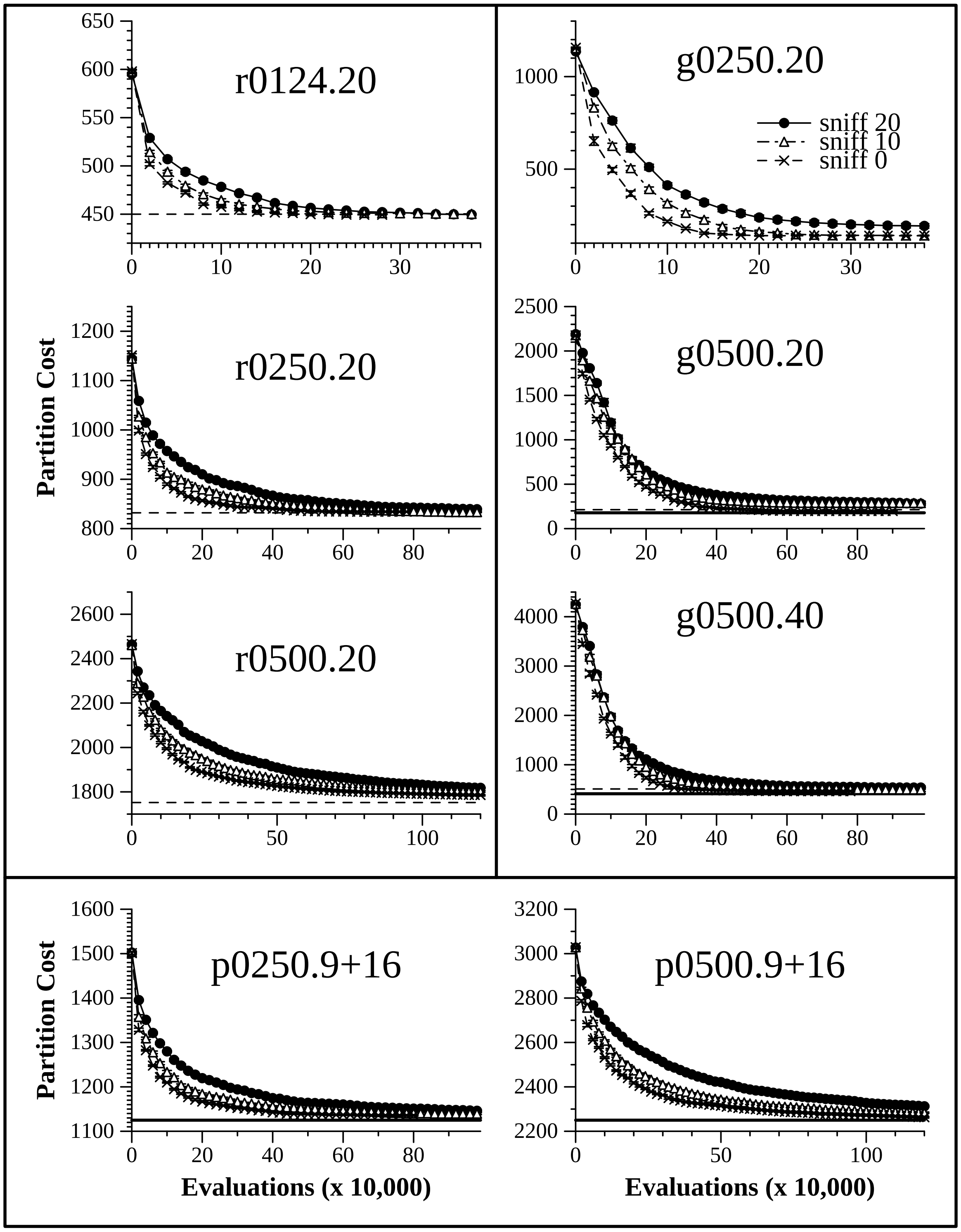

Figure IV.26: Sniff length comparison using LS potential-based LS selection

Effect of LS Selection Methods

As each of the three selection methods does best with a sniff length of zero, we compare their performance using this value. Figure IV.27 shows that there is virtually no difference between the methods in terms of the final solutions found. Where a clear difference does show up, however, is in the early stages of the runs. On all graphs, fitness-based LS selection makes the fastest progress. This is as expected, since this method exploits the best members of the population with local search to the exclusion of others. In early stages, it is therefore likely to have solutions which have undergone more LS than any solutions under the other methods. For this same reason, it is somewhat surprising that this method is competitive with the others in terms of final solutions. With its self-amplifying focus on the best solutions, one might expect it to lose diversity early on and be unable to continue searching effectively. Apparently this is not the case.

That the LS potential method is no better than the others counters our original expectations (see Section III.C.3). The results indicate that our efforts to allocate LS intelligently are no more helpful than simply allocating it randomly. There are two possible explanations for this: the measure we use for LS potential is poor, or biasing LS selection towards solutions with high LS potential is unhelpful.

Both of these hypotheses are testable to some extent. An EA+LS can be run in which in addition to the usual LS, "off-line" LS is run to completion on every solution. The results of these additional local searches do not influence the algorithm, but rather are used for analysis purposes. In particular, this would provide us with the true LS potential; the difference between a solution's current fitness and the fitness of its local minimum is its potential for improvement. Hence we could directly measure the correlation between LS potential and our estimate of it. To test the second hypothesis, we could use the directly measured LS potential (as opposed to our estimate) to bias LS selection. If this method still does no better than random LS selection, then we can conclude that our intuitions are indeed flawed in some way. Both of these experiments are directions for future research.

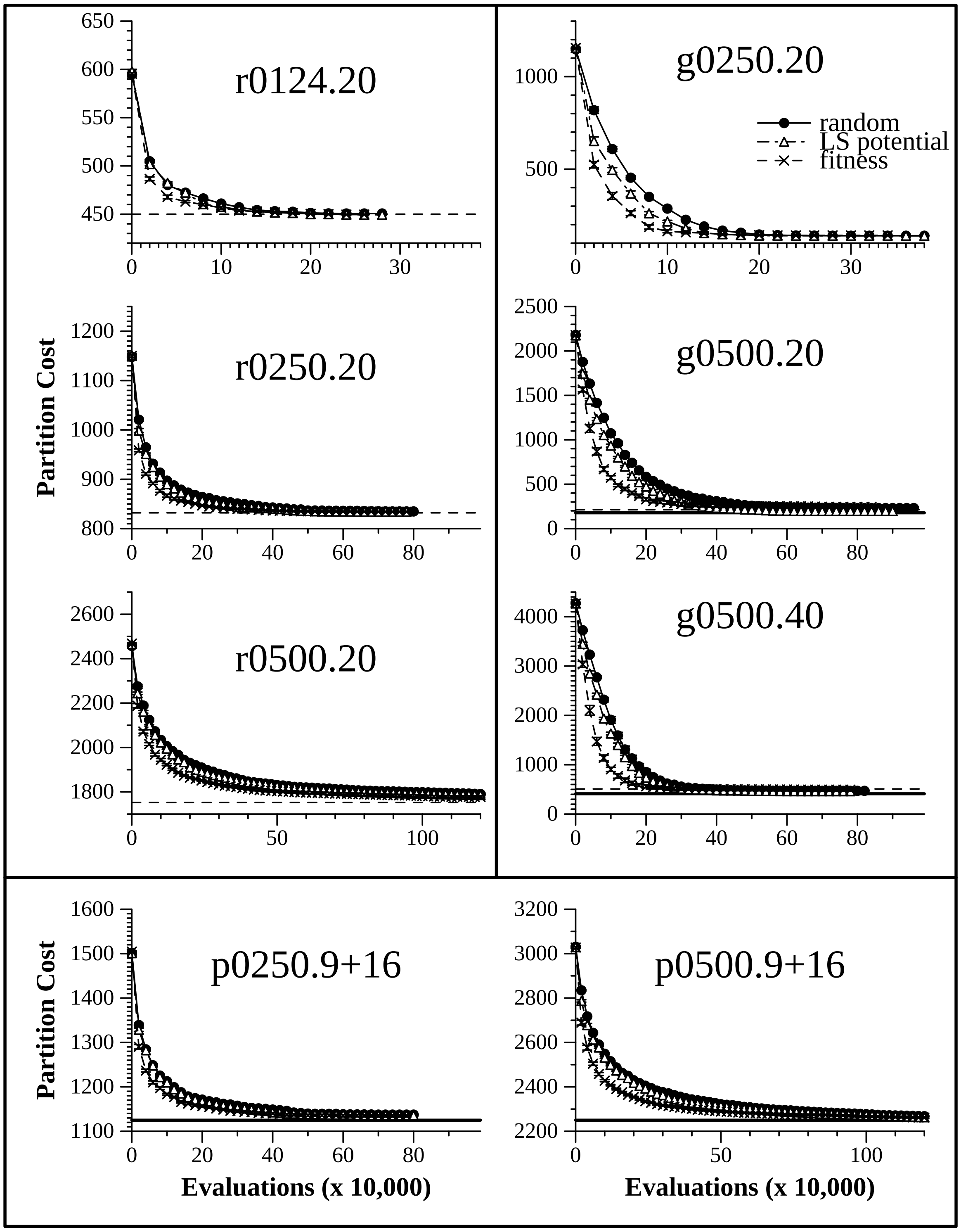

Figure IV.27: Comparison of LS selection methods for sniff length 0

In summary, we find no difference between the various LS selection methods in terms of final solution fitness. The fitness-based method is best in the early stages of the runs, which agrees with our intuitions that it is the most exploitative of the methods. For all methods the use of a sniff length is generally harmful. The fact that LS potential is no better than random selection, and that the use of sniff length is unhelpful even when using LS potential runs counter to our intuitions, but possible explanations present themselves which are testable.

IV.D.7 Lamarckian vs. Darwinian Evolution

All EA+LS experiments so far described in this dissertation have used Lamarckian evolution, in which the solutions resulting from local search replace the corresponding preoptimized solutions in the population. Hence, local improvements directly affect the genetic information in the population. An alternative to this is Darwinian evolution, in which the fitness resulting from local search is used for selection purposes, but the resulting solution itself is discarded. In this section we examine the Darwinian alternative under various mutation sizes and types of crossover. We will often refer back to Sections IV.D.3 and IV.D.4, which present the corresponding Lamarckian experiments.

Effect of Mutation Size

Figure IV.28 compares Darwinian and Lamarckian evolution for an EA using our standard crossover, but *without* mutation. Overall the Darwinian alternative is seen to result in better final solutions. This can be explained by recalling the results of Section IV.D.3, where we examined the effect of mutation size in the context of Lamarckianism. There we found that large mutations are required to prevent the search from becoming trapped in local basins, especially for the geometric graphs. In the Darwinian case, however, there is no danger of getting trapped, since the results of local are not copied back onto the genotype.

For comparison, Figure IV.29 shows the corresponding data for runs using

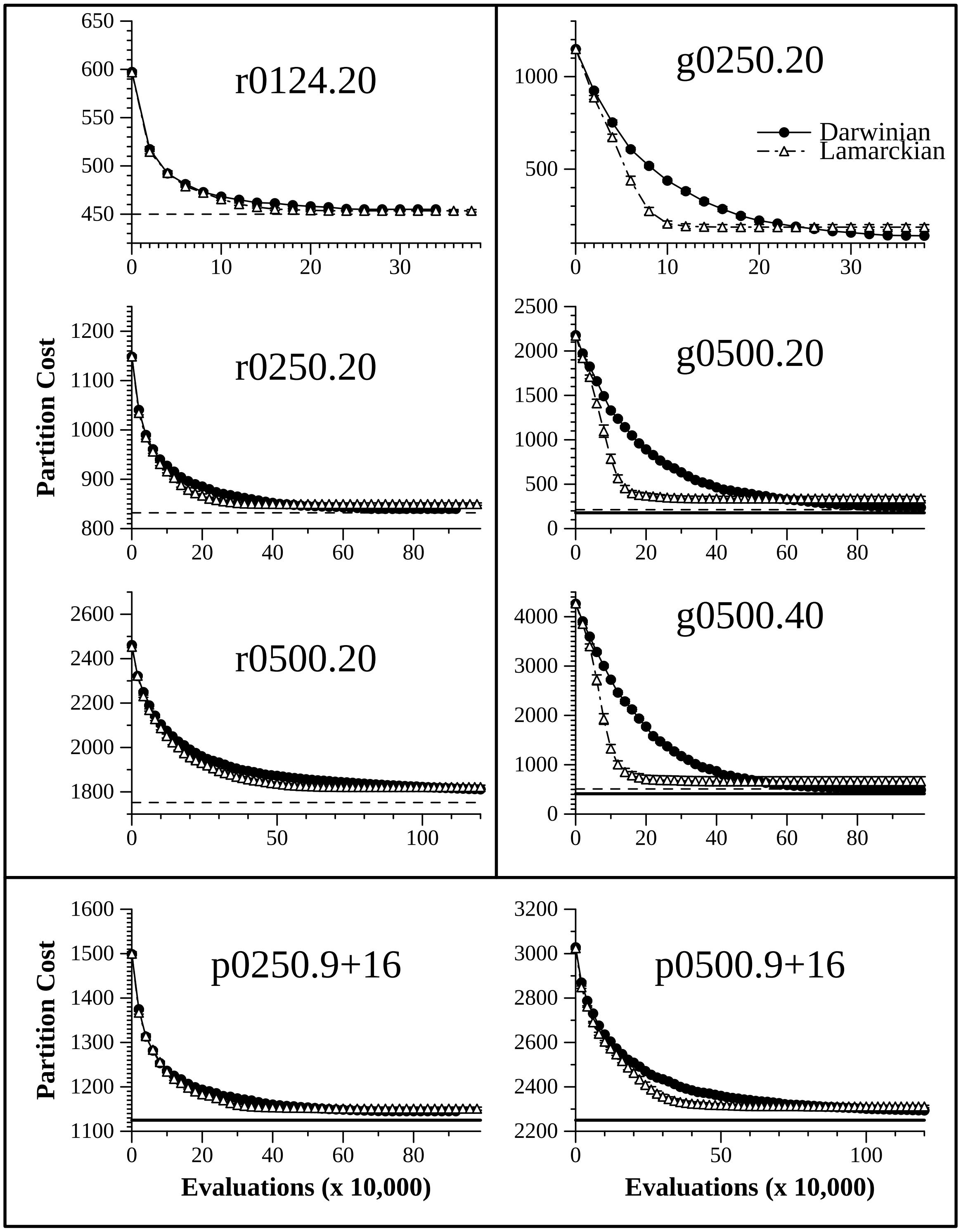

Figure IV.28: Darwinian vs Lamarckian for mutation size 0

an expected mutation size of one node swap per solution. Here we see that on the random and planted bisection graphs, where local basins are very small, the advantage of Darwinian evolution is lost entirely. This is consistent with our explanation, as even small mutations are adequate to escape most local basins on these graphs. The figure shows that on the geometric graphs, where the basins are larger, the Darwinian method is still superior. Since the genotypes in the Darwinian case are not fixated on the local minima, a solution may move between basins by a succession of small mutations, even if the basins are large.

Finally, Figure IV.30 compares Lamarckian and Darwinian evolution for a mutation size of five. We see that the Lamarckian version performs as well as or better than Darwinian on all graphs. Mutations this large are enough for the EA to escape local basins even on geometric graphs, as confirmed by the results of Section IV.D.3.

As mutation size plays an important role in the effectiveness of the Lamarckian EA+LS, the question naturally arises as to what effect it has in the Darwinian case. As noted above, mutation is no longer necessitated by the need to escape local basins. It may still be useful for some other purpose, however. Figure IV.31 compares three mutation sizes (zero, one, and five expected node swaps per solution) under Darwinian evolution. There is virtually no difference in performance of the searches. Apparently, the extra genetic variation introduced by large mutations is of no benefit or detriment to the global search. Conversely, even the lack of mutation is not detrimental. As genetic variation is clearly necessary for search to proceed, it must be the case that crossover alone is a sufficient source of this variation.

Role of Crossover

In Section IV.D.4 we saw evidence that crossover does not operate as expected under Lamarckian evolution. Specifically, it does not effectively combine building blocks from different solutions. This is despite the evidence in Section IV.C.2 that crossover *does* work as expected when local search is not being used. Our interpretation of this is that local search obviates the need for crossover by discovering building

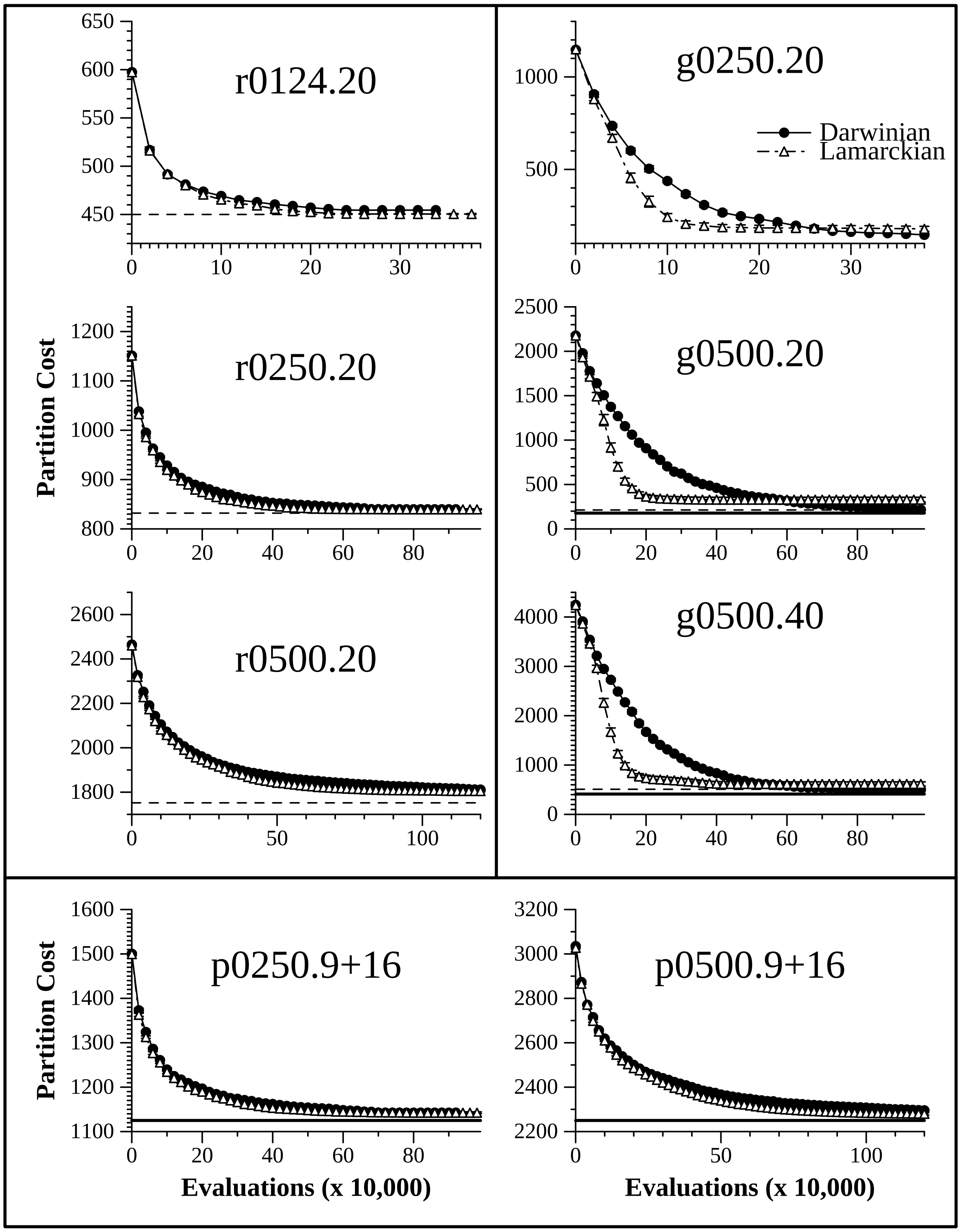

Figure IV.29: Darwinian vs Lamarckian for mutation size 1/N

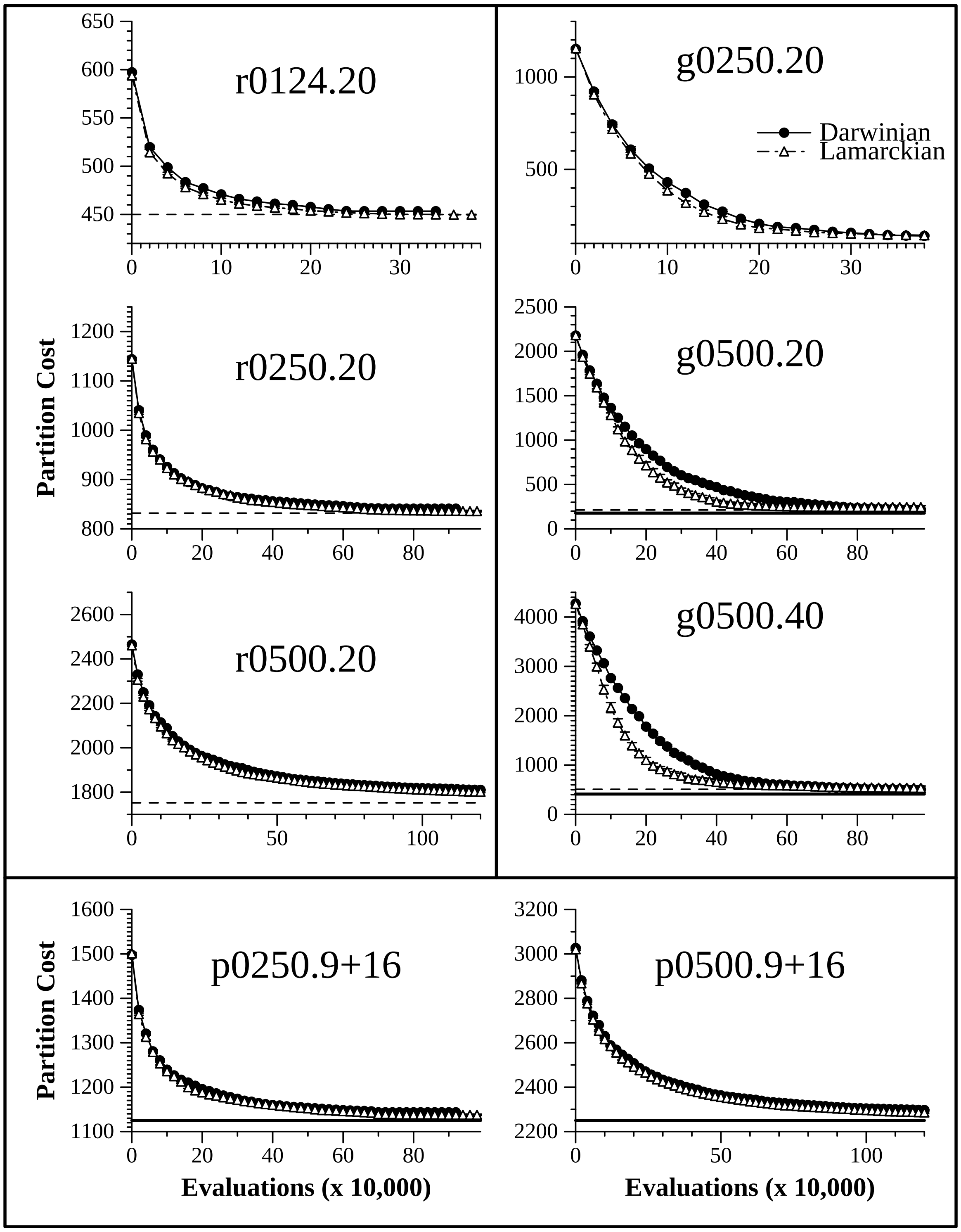

Figure IV.30: Darwinian vs Lamarckian for mutation size 5/N

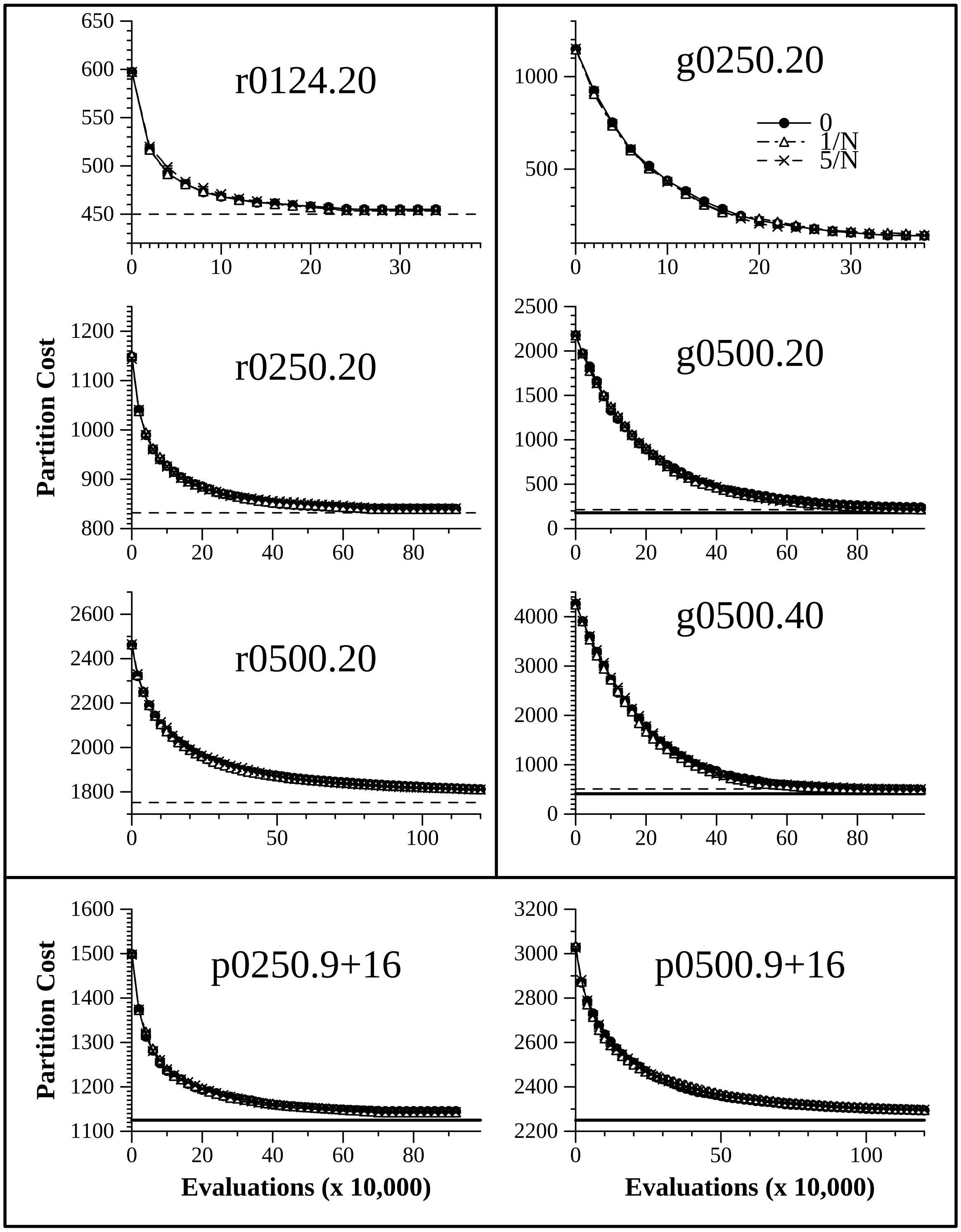

Figure IV.31: Comparison of mutation size under Darwinian evolution

blocks itself. Roughly, each application of local search find all composable subsolutions.

In light of the Lamarckian and no-LS results, we may predict what effect crossover has in the Darwinian case. Since LS finds all building blocks each time it is applied, we still do not expect crossover to be helpful. It makes little difference that the building blocks are not encoded onto the genotype; they will be found again the next time LS is applied. Note that this argument assumes an ideal situation in which every local search finds every composable subsolution. The reality of the situation may be more complicated, but as long as most local searches find most of the building blocks, the expected function of crossover is unnecessary.

Figure IV.32 compares standard-, random-, and no-crossover (as defined in Section IV.C.2) for a Darwinian EA+LS with mutation size five. There is virtually no difference between these three methods, indicating that crossover is indeed ineffective at combining building blocks. Additionally, there is no apparent benefit or detriment to using random crossover when compared to no crossover at all. Compare this to Figure IV.31, which shows no difference in using various mutation sizes under standard crossover. These data indicate that as long as there is *some* method to introduce variation it does not matter how much is introduced, at least up to the random crossover benchmark. Since local search does not directly fixate the population on specific bit-patterns, any amount of mutation will maintain sufficient diversity.

IV.E Other Global Search Algorithms

In this section we consider global search methods other than the EA+LS. Section IV.E.1 compares simulated annealing to the EA+LS, and shows that the type of graph being searched affects performance. Section IV.E.1 analyzes the behavior of SA by performing "off-line LS" during the run. Finally, Section IV.E.2 compares the Go-With-the-Winners algorithm to EA+LS. As discussed in Chapter I, both SA and GWW have global and local aspects to their search. Since they integrate these

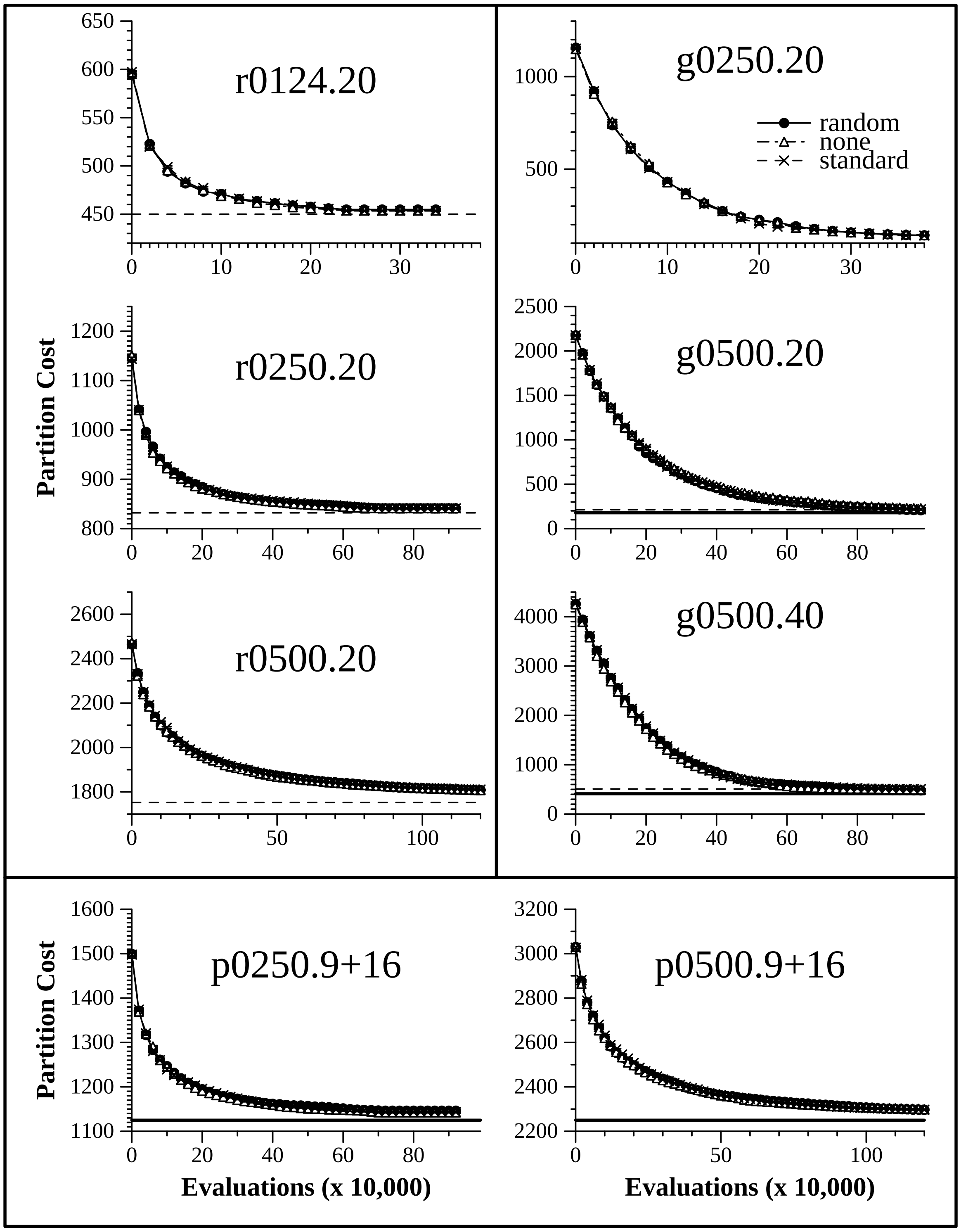

Figure IV.32: Comparison of crossover methods under Darwinian evolution

aspects less explicitly than the EA+LS, the comparisons are important.

IV.E.1 Simulated Annealing

Comparison to EA

The best known general-purpose method for graph bisection is simulated annealing [15, 52]. Figure IV.33 compares SA to the EA+LS. The EA+LS used is the generational EA+LS described in Section IV.D.1, which finds the best solutions of all the EA+LS variants considered in this dissertation. Our implementation of SA is taken from Johnson et al. [44], and is described in Section II.A.4.

We see that in all cases the EA+LS starts out finding substantially better solutions than SA. This is a direct results of its use of LS: it quickly finds local minima while SA is exploring globally. On the planted bisection and larger random graphs, SA eventually catches up with EA+LS and finds better solutions. On the geometric graphs, however, EA+LS retains it superiority throughout the runs. Since the differences at the end of the runs are too small to be seen in the figure, we display the average final solution values in Table IV.24.

Table IV.24: **Average final solutions for SA and EA+LS:** The average of the final solutions from SA and EA+LS, for the eight graph instances. Ten EA+LS runs were done for each graph, and from 30 to 70 SA runs. The number of SA runs done is also shown. A * in the *significance* column indicates that the difference between SA and EA+LS is statistically significant (two-tailed Student's t-test with $p < 0.05$).

			number	
	EA+LS	SA	SA runs	significance
r0124.20	450.4	449.9	50	
r0250.20	837.8	830.5	70	*
r0500.20	1789.8	1752.8	40	*
g0250.20	141.7	166.3	30	*
g0500.20	206.6	209.4	40	
g0500.40	464.0	468.4	40	
p0250.9+16	1141.3	1132.3	70	*
p0500.9+16	2277.0	2231.0	40	*

Note that the EA+LS runs shown here are given only as many evaluations

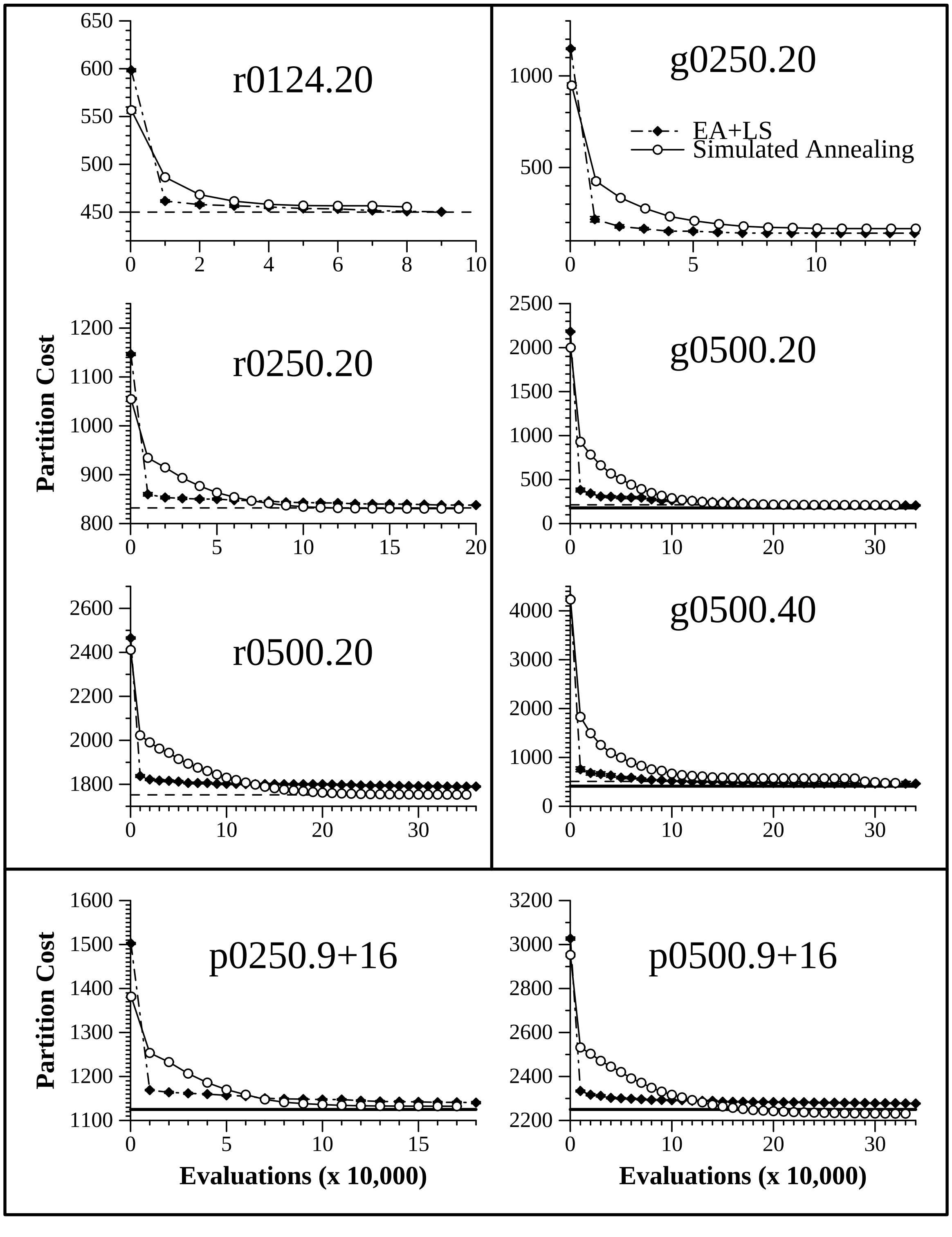

Figure IV.33: Comparison of Simulated Annealing and the EA+LS

as SA uses, and hence are shorter than we typically use. For many of these graphs, the EA+LS will continue to improve its solutions if it is allowed to run longer. SA, on the other hand, makes no further improvement if run longer, as the temperature becomes too low for any search to occur.

Basin-Finding Behavior

In order to better understand the behavior of simulated annealing for graph partitioning, we perform several runs using "off-line" local search. Each time the SA algorithm moves to a new partition, we do a complete local search and record the resulting local optimum. This is done extraneously to SA itself, so that the current partition and its cost are unmodified. In this way we get a sense of the trajectory of the search, how frequently it changes local basins, and how often it finds new basins.

Common intuition about simulated annealing suggests roughly what to expect. At high temperature, SA will visit many local basins without spending too much time in any one basin. As the temperature is lowered, the frequency with which SA explores new basins should decrease. Eventually, it becomes confined to smaller and smaller groups of basins, until finally the temperature is so low that it cannot escape the basin it is in. The experiments described here confirm this intuition on some graphs, and help to characterize the type of exploration which occurs in the later stages of a run. We will see, however, that for many graphs (especially sparse ones) the intuition isn't quite right. The reason has to do with nodes of low affinity (see Section IV.A.2).

In order to maintain consistency in our determination of local basins, a deterministic local searcher is used for these experiments. A randomized local searcher would complicate the analysis by reporting different local minima for adjacent solutions (or even the same solution). In particular, we use a steepest descent strategy over the unbalanced neighborhood, in which the nodes are ordered beforehand so as to break ties (multiple equally good "best" moves) during the local search.

Figure IV.34 shows the progression of a typical SA run on the graph r0124.20.

As a function of evaluations, this graph displays the cost of the SA's solution at each step and the cost of the local optimum associated with each. As expected, SA's average solution gets gradually better as the temperature decreases. One thing to note, though, is how volatile it is. The cost of the current solution repeatedly increases and decreases by as much as 50% of its entire range. Despite this, the solutions are of relatively high quality: throughout virtually the entire run no partition is visited whose cost is greater than 90% of the average random partition cost.

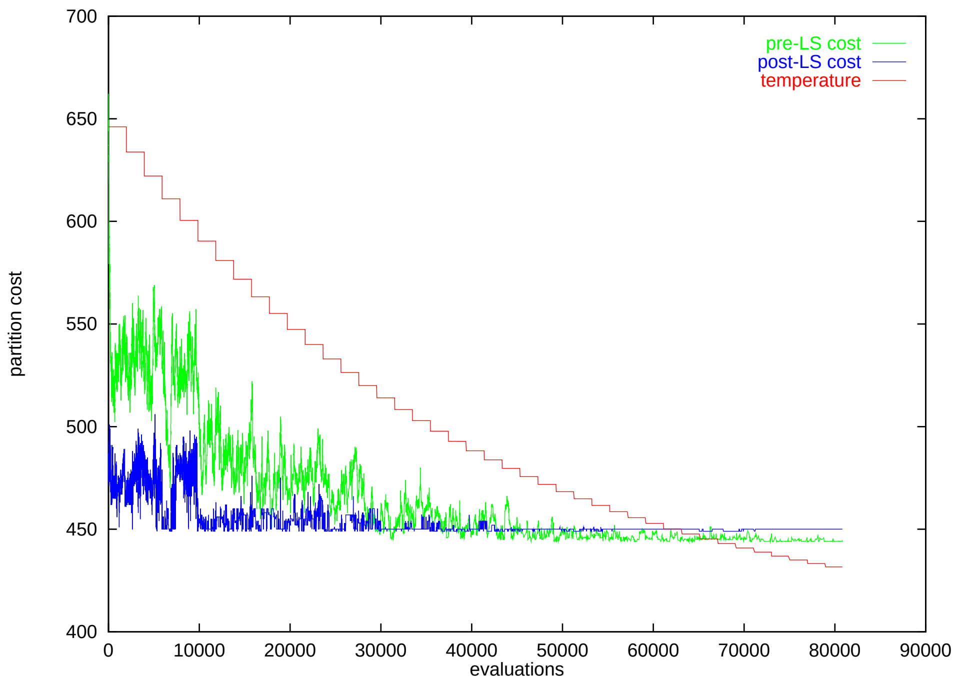

Figure IV.34: **SA with off-line LS on r0124.20:** The progression of a single SA run. The light-colored curve shows the partition cost of SA's solution at each step. The darker (bottom) curve shows the cost of the local optima associated with SA's solutions. Finally, the uppermost curve shows SA's temperature (scaled to fit the plot). For comparison, a random partition has expected cost 639.3.

As the temperature is lowered, the volatility of the solution cost decreases, and SA's solutions are closer in cost to their local optima. This corresponds to the intuition that SA settles down and becomes more like local search as it "cools." The value of the corresponding local optima is also volatile. In fact, good basins are found early on—LS finds solutions within the first 1,000 evaluations which are as good as the final solution, although SA doesn't find such solutions until much later—and lost repeatedly.

Another interesting feature of this figure is that SA starts finding partitions which are *better* than their corresponding local optima. This is because SA searches over unbalanced partitions, whereas the local optima are all balanced. In this graph it is possible to obtain a lower partition cost (including the imbalance penalty) by being slightly out of balance. Furthermore, the best unbalanced partitions found by SA do not correspond to the best balanced local optima. In the final stage of the run shown SA spends most of its time in a region whose local optimum is worse than that found previously (450 vs. 449), but which nevertheless has better unbalanced partitions. In at least this graph, then, allowing unbalanced partitions with the penalty can mislead the search.

To better understand how the SA search progresses, we can also look at how frequently SA changes basins and how often it finds *new*, previously unvisited, basins. Figure IV.35 shows the percentage of moves made by SA which change local basins (alternatively, for which the associated local optimum changes), as the SA run progresses. We see that this percentage gradually decreases on average, with SA switching basins only rarely after 60,000 evaluations. Figure IV.36 shows the percentage of moves which discover new basins. Such discoveries are rare after 45,000 evaluations, while SA is still actively switching among previously found basins.

Note that it is not the case that SA has simply explored all possible basins by this point. In all, this run visited 1,646 distinct basins. In contrast, performing local search from 10,000 random partitions found 9,265 distinct local minima. The data therefore imply that SA gradually settles into a narrow region, and eventually into a single basin, matching our intuition.

Unexpected Behavior In contrast to r0124.20, there are several graphs for which SA does not behave as originally expected, in terms of the frequency of changing local basins and discovering new basins. These graphs include those with very low degree, having many degree zero nodes, and to a lesser extent more dense graphs for which

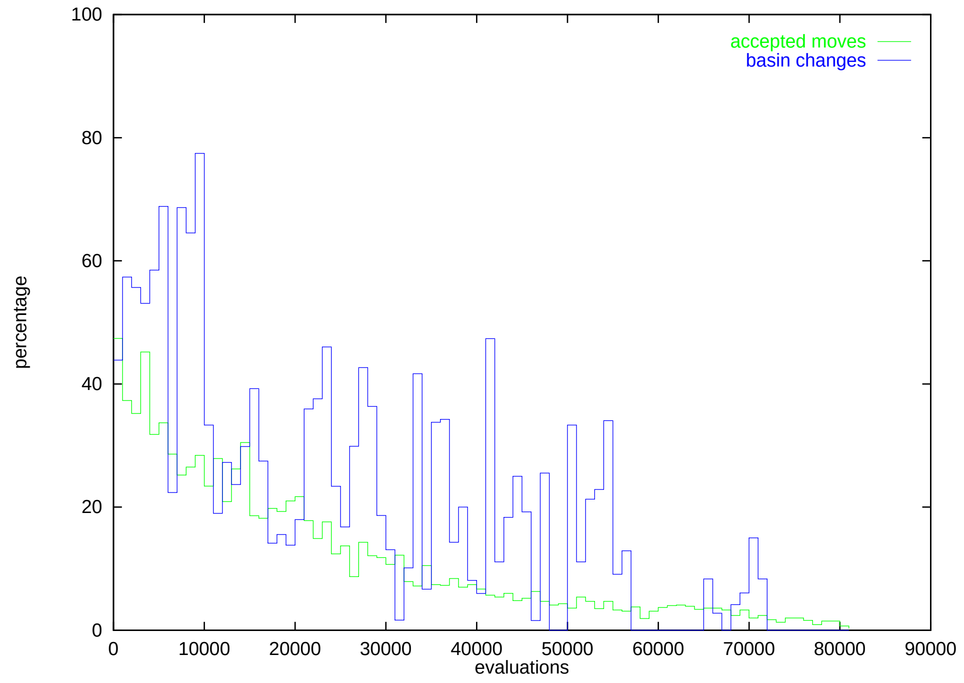

Figure IV.35: **Frequency of local basin changes on r0124.20:** For the same SA run as in Figure IV.34, the percentage of moves which are accepted, and the percentage of the accepted moves which change local basins. The data are aggregated into blocks of 1,000 SA evaluations each to generate the bar graph.

low affinity nodes are common.

Figure IV.37 shows the progression of a typical run on the graph r0124.03. As with r0124.20, the solution gradually gets better but varies greatly in the process: the cost of the current solution repeatedly increases and decreases by as much as 50% of its entire range. As before, the solutions are of relatively high quality: generally staying below 75% of the average random partition cost.

An interesting feature of this graphs is the sharp decrease in cost at around 35,000 evaluations. What apparently happened here is that SA found a series of improvements which were too good to be undone given the temperature at that point (alternatively, the volatility was too low to overcome the large gains). As seen, this event coincided with an improvement in the local optimum as well, indicating that SA found a new better basin. Other than this one event, however, note that the quality of the local optima does not follow a clear trend as the run progresses. The

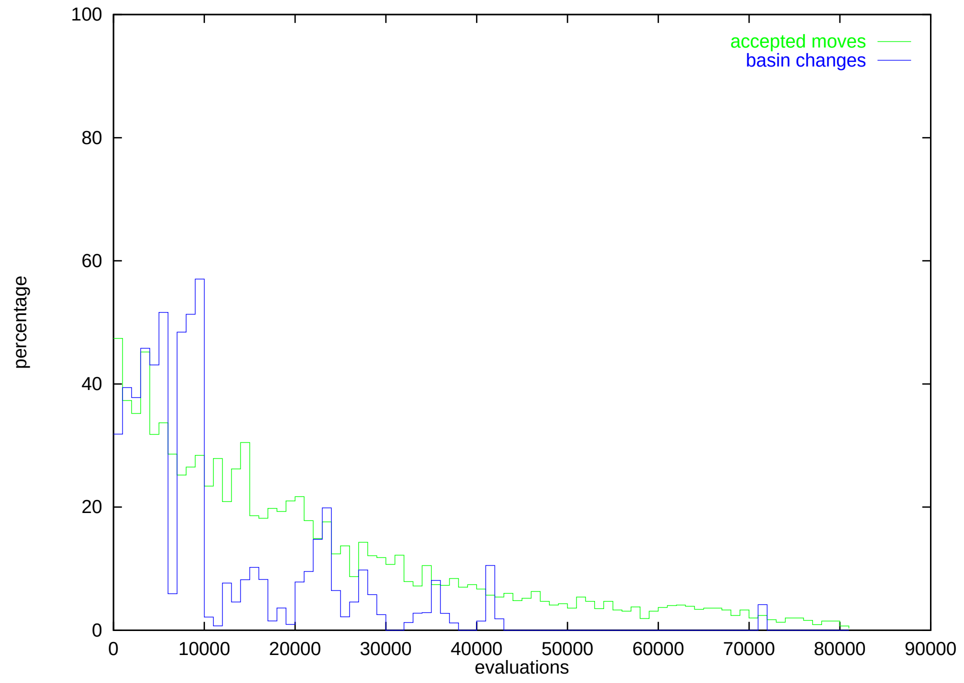

Figure IV.36: **Frequency of new local basin discoveries on r0124.20:** For the same SA run as in Figure IV.34, the percentage of moves which are accepted, and the percentage of the accepted moves which enter new local basins. The data are aggregated into blocks of 1,000 SA evaluations each to generate the bar graph.

values of the optima go up and down, and in fact very good local basins (optimum with value 15 versus the final value of 13) are found early on but are then lost.

More interesting are the plots showing the frequency of switching basins (Figure IV.38) and finding new basins (Figure IV.39). Both of these frequencies *increase* throughout the run, averaging over 70% by the end of the run. This behavior continues even after all the important activity has ceased. After 80,000 evaluations, there is no improvement in in the SA's solution cost or the costs of the associated local minima. Note that the absolute rates of these events (basin switching and basin discovery) in terms of evaluations is not increasing. What is increasing is the percentage of SA's accepted moves which switch or discover basins. This is a rather surprising fact: on this graph *SA's move become more likely to switch (and discover) basins as the temperature cools!*

Since SA flips only a single bit each move, it takes several accepting moves

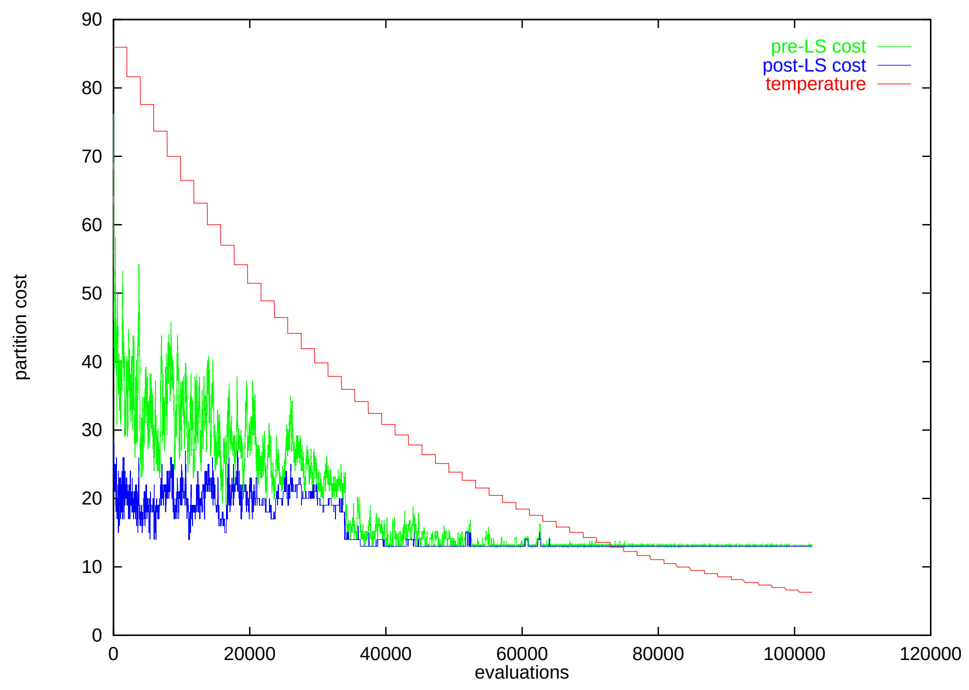

Figure IV.37: **SA with off-line LS on r0124.03:** The progression of a single SA run. The light-colored curve shows the partition cost of SA's solution at each step. The darker (bottom) curve shows the cost of the local optima associated with SA's solutions. Finally, the uppermost curve shows SA's temperature (scaled to fit the plot). For comparison, a totally random partition has expected cost 75.1.

for it to significantly change its current solution. Since over 70% of its moves in the later stages are to new local basins, this implies that this graph must have very small basins. Indeed it does; this graphs has 12 nodes of degree zero, which leads to many equivalent partitions packed together with only small barriers between them (recall Section IV.A.2). SA is simply wandering around such a region sampling the various equivalent partitions.

Given that SA is actively sampling new basins long after it has ceased improving, we may ask how large a region of the overall search space it spans during the late stages. SA changed basins 407 times between evaluation 80,000 and the end of the run (over 100,000 evaluations). The 407 associated local minima show a strong similarity to each other. Of the 124 nodes in the graph, 107 are labeled (set 0 or 1) the same way by every one of the minima. These nodes are apparently "fixed" and cannot be moved once the temperature gets low. Of the 17 nodes which do get

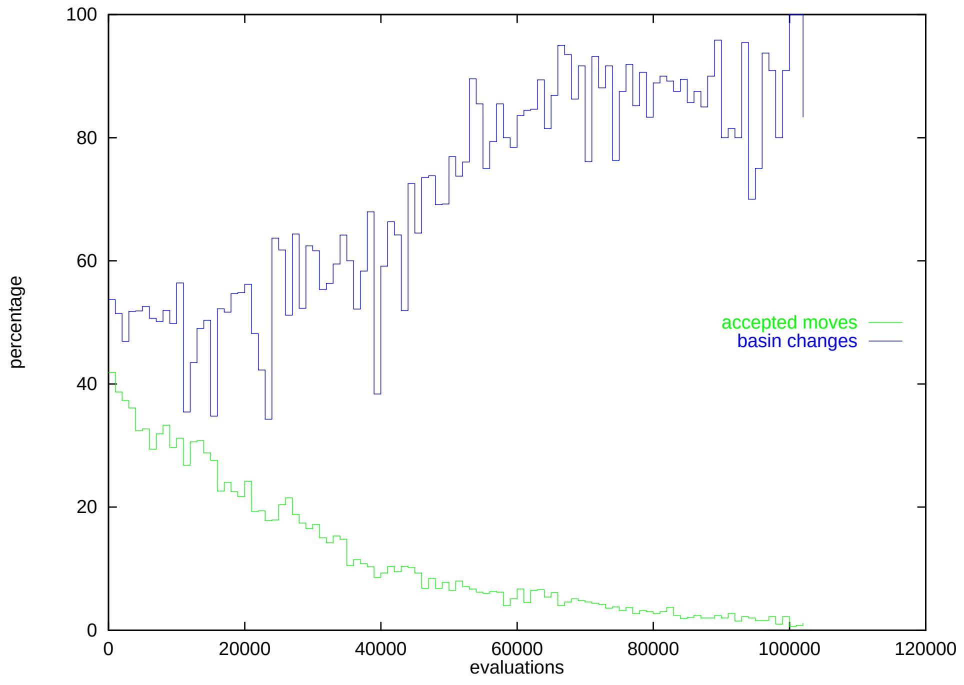

Figure IV.38: **Frequency of local basin changes on r0124.03:** For the same SA run as in Figure IV.37, the percentage of moves which are accepted, and the percentage of the accepted moves which change local basins. The data are aggregated into blocks of 1,000 SA evaluations each to generate the bar graph.

moved, 12 are the degree zero nodes mentioned earlier. The other five all have degree two, but have an affinity of zero (cf. Section IV.A.2[5]). This allows them to be moved freely without modifying the partition cost.

The above comments are for a single SA run on r0124.03. The qualitative observations about increasing basin switching and discovery are repeatable across runs, as is the observation of a small region of the space being explored in the final stage. In particular, there is a core set of approximately 100 nodes which are fixed in the late stages of any run, though they may be set to different values in different runs.

Denser Graphs Even in graphs which have no nodes of degree zero, SA may continue to explore different basins throughout the run. Figures IV.40 and IV.41 show the frequency of basin changes for the graphs r0500.10 and r0500.20, respectively. We

[5]In fact, these five nodes are the basis for the examples of zero affinity nodes in Section IV.A.2.

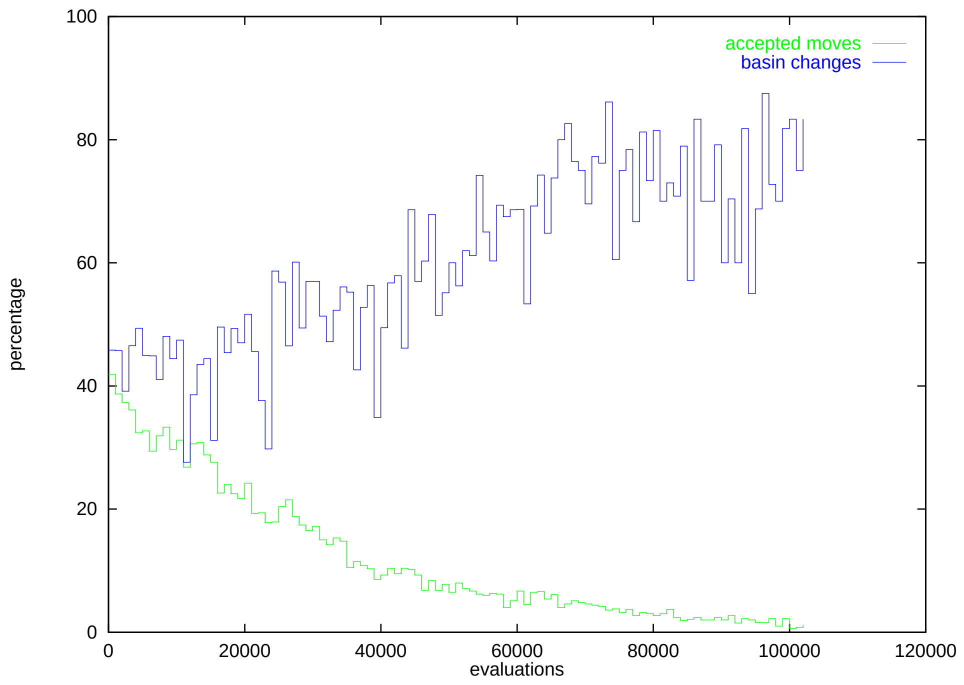

Figure IV.39: **Frequency of new local basin discoveries on r0124.03:** For the same SA run as in Figure IV.37, the percentage of moves which are accepted, and the percentage of the accepted moves which enter new local basins. The data are aggregated into blocks of 1,000 SA evaluations each to generate the bar graph.

see that the frequency increases in the latter stages despite the fact that these graphs have no nodes of degree zero (the lowest degree is 2 for r0500.10 and 7 for r0500.20). Note that this effect is not as pronounced as in the sparse graph r0124.03; in particular the frequency decreases for roughly the first half of the runs, before starting to increase again.

If we examine which nodes are being moved by SA at low temperature, we again see that the vast majority are fixed in place. Specifically, for the r0500.10 run, only 43 of the 500 nodes move after the evaluation 300,000. Similarly, only 35 of the nodes in r0500.20 are moved after evaluation 300,000. It is instructive to classify these nodes according to how much time each spends on either side of the partition. Call a node “fixed” if it stays on one side the entire time. Call it “semifixed” if it spends more than 75% of it times on one side. Otherwise it is “loose.” Table IV.25 shows the number of each of the types for r0124.03, r0500.10, and r0500.20, as well

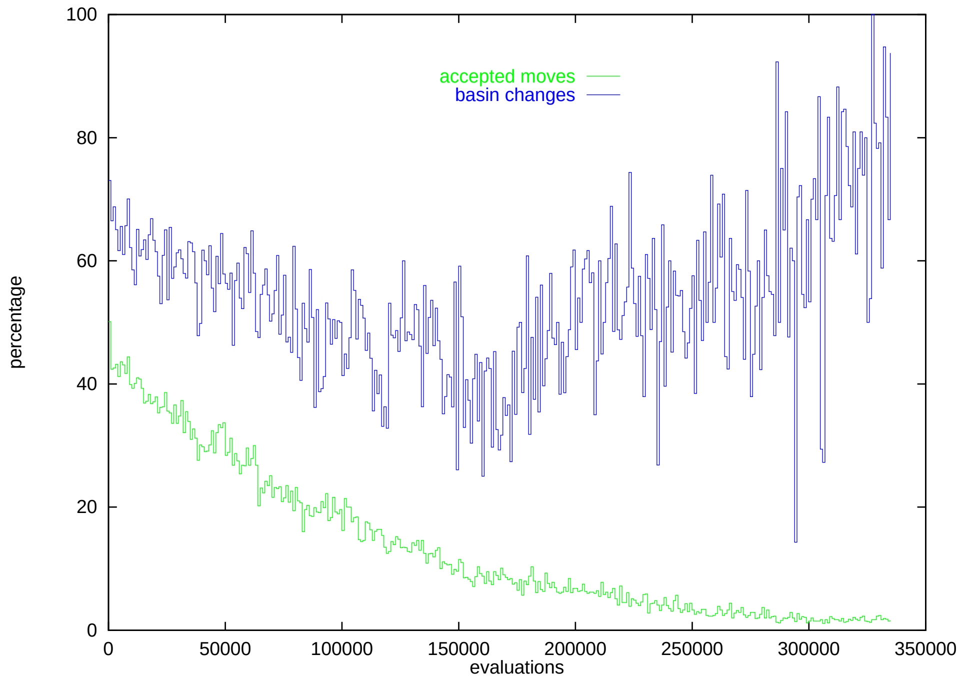

Figure IV.40: **Frequency of local basin changes on r0500.10:** The percentage of moves which are accepted, and the percentage of the accepted moves which change local basins. The data are aggregated into blocks of 1,000 SA evaluations each to generate the bar graph.

as the average affinity for nodes of each type. We see that as a group the fixed nodes have the highest average affinity, and the loose nodes have the lowest.

In summary, we have examined some intuitions about how simulated annealing operates, specifically that it should change basins and discover new basins less frequently as the temperature is lowered. This intuition turns out not to be quite true for some graphs. The reason has to do with equivalent or nearly equivalent partitions which differ by nodes of low affinity. Although SA may continue to discover new basins throughout the run, it does become confined to a tiny fraction of the search space eventually.

IV.E.2 Go-With-the-Winners

The Go-With-the-Winners algorithm of Dimitriou and Impagliazzo [19] makes for another interesting comparison of global search algorithms. As described in Sec-

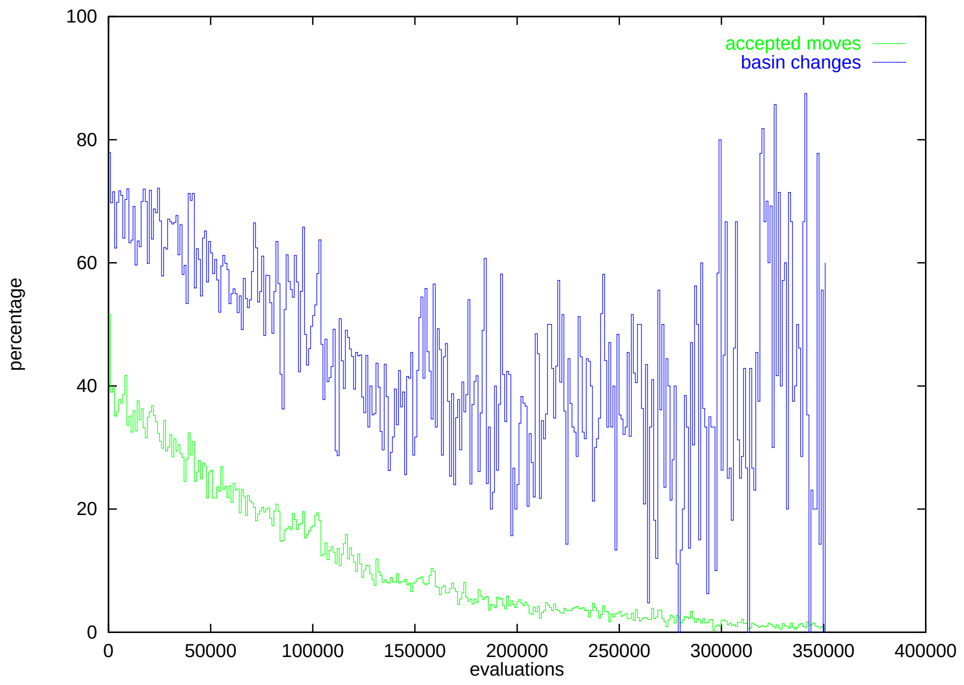

Figure IV.41: **Frequency of local basin changes on r0500.20:** The percentage of moves which are accepted, and the percentage of the accepted moves which change local basins. The data are aggregated into blocks of 1,000 SA evaluations each to generate the bar graph.

tion I.A.3, GWW attempts to maintain uniform sampling of all solutions which are better a given *threshold*. This threshold is decreased by one at each stage of the algorithm, so that the solutions in the population continually get better. GWW excels at *exploratory search* [14], but can also be used as an optimization method by setting its algorithmic parameters (number of "particles" and random walk length) small enough.

Figure IV.42 compares GWW to the EA+LS on two graph bisection instances. The GWW data in the figure is provided by Carson [12] for runs using ten particles and a random walk length of ten steps. These parameter settings were chosen for efficient optimization (as opposed to exploratory search), though minimal effort was invested in tuning them. The EA+LS algorithm we use is the generic generational EA+LS from Section IV.D.1 (and the same one we compare against SA in Section IV.E.1).

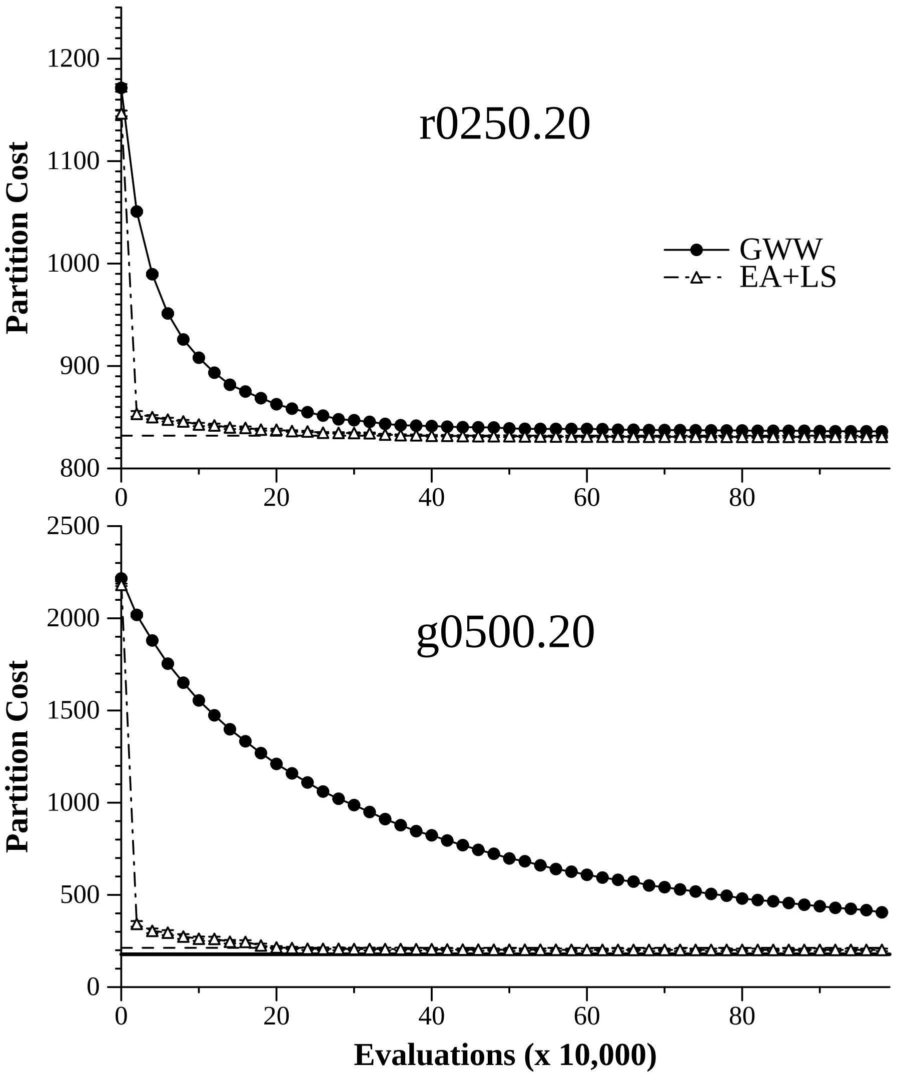

Figure IV.42: **Comparison of Go-With-the-Winners and the EA+LS:** Each method is run ten times on each graph, and standard error bars are shown.

Table IV.25: **Node affinities at end of SA run:** For three graphs, a characterization of the node movement in the final stage of a SA run. The number of nodes which are fixed, semifixed, and loose is shown, as well as the average affinity for the nodes in each group. The average is taken over the local optima associated with SA's solutions in the final stage of the run. The final stage is defined to begin at 80,000 evaluations for r0124.03, and 300,000 evaluations for the the two larger graphs.

	r0124.03		r0500.10		r0500.20	
	number	avg. affinity	number	avg. affinity	number	avg. affinity
fixed	107	2.29	457	4.60	465	6.75
semifixed	2	0.39	28	1.67	25	1.82
loose	15	0.02	15	0.39	10	1.12

We see that the EA+LS is much more efficient early on; this is a direct result of its use of LS. GWW steadily improves, however, and by 1,000,000 evaluations does nearly as well on r0250.20, though there is still a statistical difference. Note that the complete GWW runs go for much longer than shown in the figure (roughly 4-7 million evaluations for r0250.20 and 70-100 million for g0500.20), and its solutions continue improving due to the ever-decreasing threshold. The *final* solutions from GWW are not statistically different from those of the EA+LS.

A final note about the shape of the GWW curves: Carson [12] points out that if partition cost is plotted against *stage number* instead of number of evaluations, the curve should be a straight line. This is because the threshold decreases by exactly one at each stage, and the solutions in the population tend to be close to the threshold. What accounts for the concave shape is that legal neighbors (with respect to the threshold) become more difficult to find as the threshold decreases, so each stage considers successively more partitions.

Chapter V

Conclusions

We conclude the dissertation with a discussion of its major contributions and general observations. For some topics we suggest directions for future research. We end with a brief summary of the issues examined.

V.A EA+LS Effective for Graph Bisection

Previous to this dissertation, the usefulness of LS in an EA context was well-established for continuous optimization. That this benefit would carry over to combinatorial optimization was not obvious, given the various differences between the two: lack of direction or gradient, large number of dimensions, difficultly of NP-complete problems, and other issues discussed in Section II.A.2. These issues lead to substantial differences in the methods used for LS and the speed with which it operates, making it difficult to generalize from results on the continuous side.

We have shown that for at least one combinatorial problem, graph bisection, LS greatly benefits the EA for a variety of instance classes. Furthermore, even though LS itself is quite powerful on this problem, we have demonstrated that the global search performed by the EA is beneficial even in the context of LS. Not only does the EA+LS perform better than Monte Carlo local search, it is competitive with the best known general-purpose method for graph bisection, simulated annealing.

V.B New EA+LS Algorithm Integrates Global and Local

Typical EA+LS hybrids (such as the generational EA+LS in SectionIV.D.1) proceed in alternating stages of global and local search. During each generation, the EA produces a new population, and local search is then performed on part of the population. The local searches are usually performed to completion (although partial local searches have been explored by Hart[38]). The specific results of local search generally are not kept from one generation to the next, though they do influence selection of individuals and, in the case of Lamarckian evolution, the genotypes themselves.

We have developed a new EA+LS variant which more tightly integrates global and local search, while still allowing the benefits of complete local searches. The primary distinguishing features of this algorithm are the use of extremely short LS lengths and the maintenance of partially optimized solutions in the population which may be more fully optimized as the algorithm progresses. This algorithm allows the global and local search components to influence each other at a finer granularity, which may allow for more effective search on some problems. In our experiments it was never more effective than the generic generational EA+LS, though this may be because of the limited range of options explored with regards to LS selection. An additional advantage of our algorithm is that allows careful experimental control over many search parameters which may be important to search effectiveness. We make extensive use of the algorithm throughout this dissertation.

V.C LS Compositionality Hypothesis

The major result emerging from the experiments is that LS obviates the need for crossover. That is, despite evidence that crossover very effectively combines building blocks when LS is *not* used, this advantage disappears when LS is used. We hypothesize that this is because LS itself is able to find the very same building blocks that crossover would otherwise combine. To our knowledge this effect (whether or not our explanation for it is correct) has not been appreciated before. If it generalizes to other problems, it may be very important to the EA community, in which the practical usefulness of crossover is sometimes hotly debated. It may be that the debate is moot in the context of LS.

While our interpretation of the effect (namely, that LS finds all the building blocks) has not been directly verified, we have explored one other potential explanation and rejected it (see Section IV.D.4). One way to verify our hypothesis would be to explicitly identify the building blocks for a particular instance. In the case of graph bisection, we suspect that small cliques are a source of building blocks, with higher fitness being associated with having the entire clique on the same side of the partition. We can test this by examining the results of successful crossovers (those that produce a child with better fitness than either parent) in an EA without LS. If all such successful crossovers combine cliques from both parents, that would be strong evidence that the small cliques indeed correspond to the building blocks. If so, then it would be a simple matter to check that LS always groups the nodes of each clique together, thereby eliminating the need for crossover. An analysis along these lines is a possible direction for future research.

It is possible that LS usurping crossover's role is specific to graph bisection. We do not know any comparison of crossover's effectiveness with and without local search for other problems. In fact, there are several studies [38, 71, 56] which independently examine both the use of LS and the effectiveness of crossover for continuous optimization, but not the effectiveness of crossover when LS is used. Another future research direction is to perform this comparison on other problems, both combinatorial and continuous.

Finally, our observations may simply be the result of the crossover operator we use. It appears that it operates on the same small substructures as LS. This may point to the need for a crossover operator which can combine "higher-order" building-blocks, analogous to the need for mutation to make larger-scale changes. It could be that the crossover operator we use (RAR_∞), is effective at combining small independent cliques, but is too disruptive to maintain large groups of cliques, which may be the appropriate higher-level building blocks. One can imagine a different recombination operator being more effective at this task, and hence being more beneficial then simple random crossover even when LS is used. Note that such an operator may *not* be effective without the use of LS, as the large structures on which it operates may take too long to be discovered through mutation and selection alone.

V.D Large Mutation Sizes and Instance-Specific Heuristics

Related to the issue of random crossover (or macromutation) is the issue of the appropriate size of mutations. We saw that very large mutations were not detrimental to the EA+LS search, but that standard mutation sizes were ineffective for some instance classes. Specifically, for problems with definite basin structures (geometric graphs), under Lamarckian evolution, small mutations are not beneficial in comparison to no mutation, but large mutations help. This is as we expected, since mutations within a basin simply get returned to the local minimum by LS. Since our expectations followed our general understanding of the EA+LS search, we expect this result to generalize to other problems, and we offer the following recipe: *when LS is used with Lamarckian evolution, the appropriate size of mutation will be at least as large as the typical basin size.*

The above rule for setting mutation size is an example of an *instance-specific heuristic*, or an algorithm tailored to specific features of the instances under consideration. For difficult problem classes (e.g. NP-complete) it may be necessary to exploit such features to improve search effectiveness. Here we have focussed on mutation size, but appropriate values for other algorithmic parameters (local/global ratio, LS length, etc.) may also depend on instance features. For that matter, some types of problems probably call algorithms other than the EA+LS. There are two parts to using instance-specific heuristics: determination of the instance features, and exploitation of these in the algorithmic design. For the former, one may perform experiments such as those in Section IV.D.3, where we carefully analyzed the basin sizes and structures through the repeated use of LS and mutation. More generally, we point to the ongoing work by Carson and Impagliazzo [14] with the Go-With-the-Winners algorithms (see Section I.A.3), which can be used to explore search space structure. We anticipate future integration of that research with our own, with the goal of identifying particular features of instances which allow informed choices as to which search algorithms are appropriate.

V.E Darwinian Evolution Competitive and Robust

An important observation from Section IV.D.7 is that Darwinian evolution performs just as well as Lamarckian. Direct comparisons in the literature are scarce, but the Lamarckian option is generally used for optimization. Our results show that this may not be a well-justified choice. The reason that Darwinian evolution works well may be related to LS compositionality hypothesis: every complete LS finds every building block. Hence, it makes no difference whether the results of LS are encoded on the genotype, as the same building blocks will be found the next time LS is applied.

Another explanation favored by Belew [7] involves the *Mastery effect*[36]. Roughly, this is the notion that evolutionary pressure to improve the genotypes is reduced once they are “good enough” for LS to take them to the global optimum (or at least the best solution seen so far by the population). Note that the potential role of the Mastery effect is somewhat diminished by our use of partial LS: since we use very small amounts of LS at a time (ten evaluations), there is an advantage to being very close (within ten steps) to the optimum.

Unlike the Lamarckian case, we saw that the Darwinian EA+LS is robust across different mutation sizes. Our explanation of this is that since LS results are not encoded on the genotype, any variation (whether by small mutations or crossover) is adequate to move from basin to basin by successive applications. Hence, unlike in the Lamarckian case, the search cannot become "stuck" in suboptimal local basins.

V.F Problems are Difficult

Finally, given that for all our problem instances, various versions of the EA+LS and SA find roughly the same partition costs, and that all runs seem to stop improving after some point, it is tempting to conclude that we have solved these instances. It appears they might be too easy for a comparison of different algorithmic parameter settings, since many settings may lead to the optimal solution. The reality, however, is that the instances examined in this dissertation are not trivial. Despite repeated applications of simulated annealing and EA+LS under a variety of configurations, the best solutions we found are substantially worse than the best known on geometric graphs. Figure V.1 gives an expanded view of the average partition costs found by EA+LS, MCLS, and SA for g0500.20, compared to the best known solution.[1] This indicates that the geometric graphs, at least, are fertile ground for the development and testing of new heuristics. The random graphs may be equally difficult, but we do not have an independent method for finding good solutions.

V.G Summary

In summary, we have shown that the EA+LS hybrid is more effective for graph bisection than either the EA or LS alone, and that it is competitive with simulated annealing. A new variant of the EA+LS has been developed which interleaves the global and local search operators. We have discussed results concerning the effectiveness of crossover and the role of mutation in the context of LS; these insights

[1]The best solution was found by Johnson [44] using an algorithm developed specifically for geometric graphs. Roughly, this algorithm scans a straight line across the graph (laid out geometrically), and chooses the best partition encountered during the scan. This partition is then further optimized by local search.

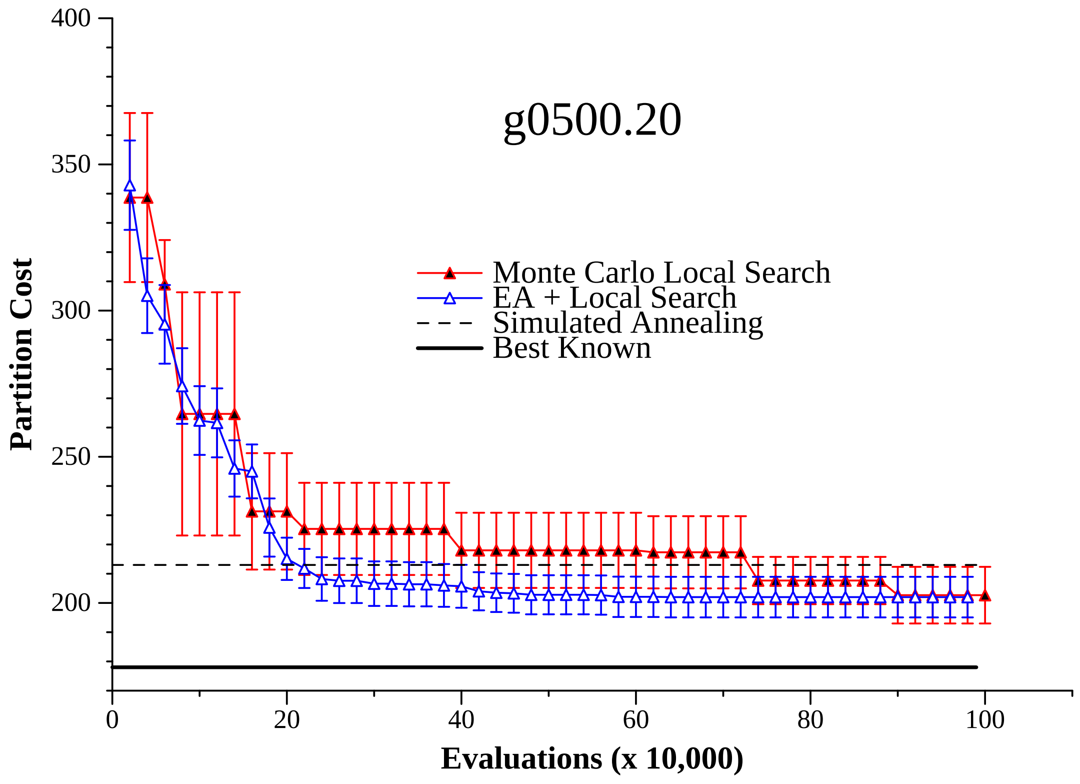

Figure V.1: **EA+LS, MCLS, and SA vs. best known:** For g0500.20, an expanded view of end of ten EA+LS runs, compared to Monte Carlo LS. The dashed horizontal line is the average SA performance, and the the solid horizontal line is the best solution known.

point to the possibility of instance-specific heuristics, in which the search algorithm is tailored to specific features of the instances under consideration. We have seen that Darwinian evolution is as effective as Lamarckian, and is more robust under changes to the genetic operators. In addition to the results mentioned in this chapter, we have also explored the effectiveness of steady-state vs. generational EAs, different local search lengths, and various local/global ratios.

Bibliography

[1] David H. Ackley. *A Connectionist Machine for Genetic Hillclimbing.* Kluwer Academic Publishers, 1987.

[2] David H. Ackley and Michael L. Littman. A case for lamarckian evolution. In Chris G. Langton, editor, *Artificial Life III*, pages 487–509. Addison-Wesley, 1994.

[3] David Aldous and Umesh Vazirani. Go with the winners. In S. Goldwasser, editor, *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 492–501. IEEE, New York, 1994.

[4] S. Arora, D. Karger, and M. Karpinski. Polynomial time approximation schemes for dense instances of NP-hard problems. In *27th Annual ACM Symposium on Theory of Computation*, pages 284–293, 1995.

[5] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, and Genetic Algorithms.* Oxford University Press, New York, 1996.

[6] Thomas Bäck, Ulrich Hammel, and hans Paul Schwefel. Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1(1):3–17, April 1997.

[7] Richard K. Belew. personal communication.

[8] Richard K. Belew, John McInerny, and Nicol N. Schraudolph. Evolving networks: Using the genetic algorithm with connectionist learning. In Chris G. Langton, Charles Taylor, J. Doyne Farmer, and Steen Rasmussen, editors, *Proceeding of the Second Conference on Artificial Life*, pages 511–548. Addison-Wesley, 1991.

[9] Kenneth D. Boese, Andrew B. Kahng, and Sudhakar Muddu. A new adaptive multi-start technique for combinatorial global optimizations. *Operations Research Letters*, 16:101–113, 1994.

[10] R. Boppana. Eigenvalues and graph bisection: An average case analysis. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, pages 280–285. IEEE, 1987.

[11] John Cairns, Julie Overbaugh, and Stephan Miller. The origin of mutants. *Nature*, 335(6186):142–145, September 8 1988.

[12] Ted Carson, October 1998. personal communication.

[13] Ted Carson and Russell Impagliazzo, September 1998. personal communication.

[14] Ted Carson and Russell Impagliazzo. A method for experimentally exploring search spaces: The case of graphs with small planted bisections. 1999. submitted to The first Workshop on Algorithm Engineering and Experimentation.

[15] V. Cerny. A thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm. *J. Optim. Theory Appl.*, 45:41–51, 1985.

[16] L. Davis, editor. *Genetic Algorithms and Simulated Annealing.* Morgan Kaufmann, Los Altos, CA, 1987.

[17] L. Davis, editor. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, 1991.

[18] J. E. Dennis and Virginia J. Torczon. Derivative-free pattern search methods for multidisciplinary design problems. In *The Fifth AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, pages 992–993, 1994.

[19] A. Dimitriou and R. Impagliazzo. Towards a rigorous analysis of local optimization algorithms. In *28th ACM Symposium on the Theory of Computing*, 1996.

[20] A. Dimitriou and R. Impagliazzo. Go-with-the-winner algorithms for graph bisection. In *SODA*, pages 510–520, 1998.

[21] D. B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* IEEE Press, 1995.

[22] D. B. Fogel, editor. *Evolutionary Computation: the Fossil Record.* IEEE Press, 1998.

[23] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence Through Simulated Evolution.* Wiley, New York, 1966.

[24] Stephanie Forrest and Melanie Mitchell. The performance of genetic algorithms on Walsh polynomials: Some anomalous results and their explanation. In Richard K. Belew and Lashon B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, page 182. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1991.

[25] Stephanie Forrest and Melanie Mitchell. Relative building-block fitness and the building-block hypothesis. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 109–126. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[26] D. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Proceedings of the Foundations of Genetic Algorithms Workshop*, Indiana, 1990.

[27] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, 1989.

[28] D.E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In John J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms.* L. Erlbaum Assoc., 1987.

[29] V. Scott Gordon and Darrell Whitley. Serial and parallel genetic algorithms as function optimizers. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 177–183. Morgan Kaufmann, 1993.

[30] Frédéric Gruau and Darrell Whitley. Adding learning to the cellular development of neural networks: Evolution and the baldwin effect. *Evolutionary Computation*, 1(3):213–233, 1993.

[31] William Noble Grundy. Genetic algorithm variants of the GSAT algorithm, 1998. in preparation.

[32] William E. Hart. personal communication.

[33] William E. Hart. Evolutionary pattern search algorithms. Technical report, Sandia National Laboratories, Sep 1995.

[34] William E. Hart. A theoretical comparison of evolutionary algorithms and simulated annealing. In *Evolutionary Programming V*, pages 147–154, Cambridge, MA, 1996. MIT Press.

[35] William E. Hart and Richard K. Belew. Optimization with genetic algorithm hybrids that use local searches. In Richard K. Belew and Melanie Mitchell, editors, *Adaptive Individuals in Evolving Populations: Models and Algorithms*, chapter 27, pages 483–496. Addison-Wesley, 1996.

[36] William E. Hart, Thomas E. Kammeyer, and Richard K. Belew. The role of development in genetic algorithms. In D. Whitley and M. Vose, editors, *Foundations of Genetic Algorithms III*, pages 315–332. Morgan Kauffman, 1994.

[37] William E. Hart, Mark Land, and Richard K. Belew. Evolutionary algorithms with local search: Hybridization issues for continuous search domains, 1998. in preparation.

[38] William Eugene Hart. *Adaptive Global Optimization with Local Search.* PhD thesis, University of California, San Diego, 1994.

[39] Bruce Hendrickson and Robert Leland. An improved spectral load balancing method. In *Proceedings of the 6th SIAM Conference on Parallel Processing for Scientific Computing*, pages 953–961. SIAM, 1993.

[40] Bruce Hendrickson and Robert Leland. A multilevel algorithm for partitioning graphs. Technical report, Sandia National Laboratories, Albuquerque, NM, October 1993.

[41] John H. Holland. *Adaptation in Natural and Artificial Systems.* The MIT Press, 1975.

[42] L. Ingber and B. Rosen. Genetic algorithms and very fast simulated reannealing—a comparison. *Mathematical and Computer Modelling*, 16:87–100, 1992.

[43] Zhai Jinhui, Yan Yingbai, Jin Guofan, and Wu Minxian. Global/local united search algorithm for global optimization. *Optik*, 108(4):161–164, 1998.

[44] David S. Johnson, Cecilia R. Aragon, Lyle A. McGeoch, and Catherine Schevon. Optimization by simulated annealing: An experimental evaluation; part I, graph partitioning. *Operations Research*, 37(6):865–892, Nov-Dec 1989.

[45] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of computer and system sciences*, 37(1):79–100, Aug 1988.

[46] Terry Jones. Crossover, macromutation, and population-based search. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 73–80. Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1995.

[47] Kenneth A. De Jong and Jayshree Sarma. Generation gaps revisited. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 19–28. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[48] R. S. Judson, M. E. Colvin, J. C. Meza, A. Huffer, and D. Gutierrez. Do intelligent configuration search techniques outperform random search for large molecules? *International Journal of Quantum Chemistry*, pages 277–290, 1992.

[49] Thomas E. Kammeyer. *Evolving Stochastic Grammars.* PhD thesis, University of California, San Diego, 1998.

[50] Thomas E. Kammeyer, Richard K. Belew, and S. Gill Williamson. Evolving compare-exchange networks using grammars. *Artificial Life*, 2(2), 1995.

[51] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(10):291–307, Feb 1970.

[52] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, May 13 1983.

[53] Scott Kirkpatrick and Bart Selman. Critical behavior in the satisfiability of random boolean expressions. *Science*, 264:1297–1301, May 27 1994.

[54] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.

[55] Mark Land and Richard K. Belew. No perfect two-state cellular automata for density classification exists. *Physical Review Letters*, 74(25):5148–5150, June 19 1995.

[56] Mark Land, John J. SIDorowich, and Richard K. Belew. Using genetic algorithms with local search for thin film metrology. In Thomas Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 537–44. Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1997.

[57] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operation Research*, 21:498–516, 1973.

[58] G. Lueker. manuscript, Princeton University, 1976.

[59] Samir W. Mahfoud. A comparison of parallel and sequential niching methods. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 136–143. Morgan Kaufmann, 1995.

[60] Samir W. Mahfoud. *Niching methods for genetic algorithms.* PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL, USA, May 1995. IlliGAL Report 95001.

[61] S.W. Mahfoud. Population size and genetic drift in fitness sharing. In L.D. Whitley and M.D. Vose, editors, *Foundations of Genetic Algorithms 3.* Morgan Kaufmann, 1995.

[62] H. Mühlenbein. Parallel genetic algorithms, population genetics and combinatorial optimization. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 416–421. Morgan Kaufmann, San Mateo, CA, 1989.

[63] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[64] C. H. Papadimitriou, A. A. Schäffer, and M. Yannakakis. On the complexity of local search. In *Proceedings of the 22nd ACM Symposium on the Theory of Computing*, pages 838–845, 1990.

[65] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[66] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization—Algorithms and Complexity*. Prentice-Hall, 1982.

[67] A. Pothen, H. Simon, and K. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal on Matrix Analysis and Applications*, 11(3):430–452, 1990.

[68] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C*, chapter 10.4, pages 408–412. Cambridge University Press, 2nd edition, 1995.

[69] Nicholas J. Radcliffe. Forma analysis and random respectful recombination. In Richard K. Belew and Lashon B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1991.

[70] Nicholas J. Radcliffe. Genetic set recombination. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[71] Christopher D. Rosin, R. Scott Halliday, William E. Hart, and Richard K. Belew. A comparison of global and local search methods in drug docking. In Thomas Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 221–8. Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1997.

[72] H. Saran and V. V. Vazirani. Finding k-cuts within twice the optimal. In *Proceedings 32nd Annual Symposium on Foundations of Computer Science*, pages 743–751. IEEE Computer Society Press, 1991.

[73] Hans-Paul Schwefel. *Evolution and Optimum Seeking*. Wiley, New York, 1995.

[74] Bart Selman, Hector Levesque, and David Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 440–446, San Jose, CA, July 1992.

[75] H. D. Simon. Partitioning of unstructured problems for parallel processing. In *Proceedings of Conference on Parallel Methods on Large Scale Structural Analysis and Physics Applications*. Pergammon Press, 1991.

[76] K. Sims. Evolving 3D morphology and behavior by competition. In R. Brooks and P. Maes, editors, *Artificial Life IV Proceedings*, pages 28–39. MIT Press, 1994.

[77] R.E. Smith, S. Forrest, and A.S. Perelson. Searching for diverse, cooperative populations with genetic algorithms. *Evolutionary Computation*, 1(2), 1993.

[78] F. J. Solis and R. J-B. Wets. Minimization by random search techniques. *Mathematical Operations Research*, 6:19–30, 1981.

[79] G. B. Sorkin. Efficient simulated annealing on fractal energy landscapes. *Algorithmica*, 6(3):367–418, 1991.

[80] Gilbert Syswerda. A study of reproduction in generational and steady-state genetic algorithms. In *Proceedings of the Foundations of Genetic Algorithms Workshop*, Indiana, 1990.

[81] Virginia Torczon. On the convergence of pattern search methods. *SIAM Journal of Optimization*, 7(1):1–25, February 1997.

[82] Virginia Torczon. Pattern search methods for nonlinear optimization. *SIAG/OPT Views-and-News*, (6):7–10, Spring 1995.

[83] N. L. Ulder, E. H. Aarts, H.-J. Bandelt, P. J. van Laarhoven, and E. Pesch. Genetic local search algorithms for the traveling salesman problem. In *Parallel Problem Soving from Nature*, pages 109–116, New York, 1990. Springer-Verlag.

[84] D. Whitley, V. S. Gordon, and K. Mathias. Lamarckian evolution, the Baldwin effect and function optimization. In Y. Davidor, H. P. Schwefel, and R. Berlin Manner, editors, *Parallel Problem Solving from Nature - PPSN III*, pages 6–15. Springer-Verlag, 1994.

[85] D. Whitley and J. Kauth. Genitor: A different genetic algorithm. In *Proceedings 4th Rocky Mountain Conference on Artificial Intelligence*, Denver, 1988.

[86] D. Whitley, K. Mathias, S. Rana, and J. Dzubera. Building better test functions. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, page 239. Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1995.