A Beginner's Introduction to Iterated Local Search*

Helena Ramalhinho Lourenço * Olivier Martin † Thomas Stützle ‡

* Dept. of Economics and Business, Universitat Pompeu Fabra
R. Trias Fargas 25-27, 08005 Barcelona, Spain
Email: helena.ramalhin@econ.upf.es

† LPTMS, Université Paris-Sud
bat. 100, F-91405 Orsay, France
Email: martino@ipno.in2p3.fr

‡ Computer Science Department, Darmstadt University of Technology
Alexanderstraße 10, D-64283 Darmstadt, Germany
Email: stuetzle@informatik.tu-darmstadt.de

1 Introduction

The importance of high performance algorithms for tackling $\mathcal{NP}$-hard optimization problems cannot be understated, and in many practical cases the most successful methods are metaheuristics. When designing a metaheuristic, it is preferable that it be simple, both conceptually and in practice. Naturally, it also must be effective, and if possible, general purpose. Iterated local search (ILS) is such a metaheuristic and the essence of the ILS metaheuristic can be given in a nut-shell: one *iteratively* builds a sequence of solutions generated by an embedded heuristic, leading to far better solutions than if one were to use repeated random trials of that heuristic. This simple idea [4] has a long history, and its rediscovery by many authors has lead to many different names for ILS like *iterated descent* [3], *large-step Markov chains* [14], *iterated Lin-Kernighan* [7], *chained local optimization* [13], or combinations of these [1] ... Readers interested in these historical developments should consult the review [8]. For us, there are two main points that make an algorithm an ILS: (i) there is a single chain that is followed; (ii) the search for better solutions occurs in a reduced space defined by the output of an embedded heuristic. In practice, local search has been the most frequently used embedded heuristic, but in fact any optimizer can be used, be-it deterministic or not.

Our purpose here is to give an accessible description of the underlying principles of ILS and a discussion of basic implementation issues. Regarding performance, ILS has lead, in spite of its conceptual simplicity, to a number of state-of-the-art results without the use of too much problem-specific knowledge.

2 Iterating a local search

General framework: We assume we are given a problem-specific approximate algorithm that we refer to as a local search (even if in fact it is not a true local search) that is implemented via a computer routine we call LocalSearch. The question we ask is "Can such an algorithm be improved by the use

*This work was partially supported by the "Metaheuristics Network", a Research Training Network funded by the Improving Human Potential programme of the CEC, grant HPRN-CT-1999-00106. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

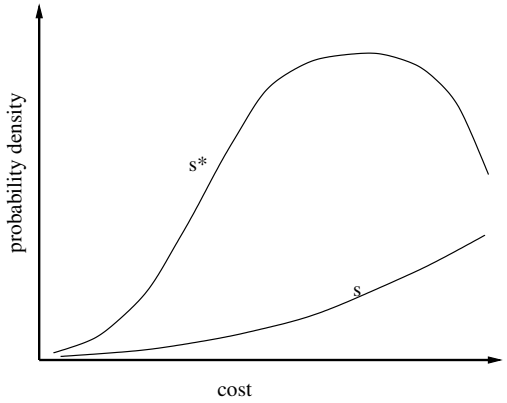

Figure 1: Probability densities of costs. The curve labeled s gives the cost density for all solutions, while the curve labeled s^* gives it for the solutions that are local optima.

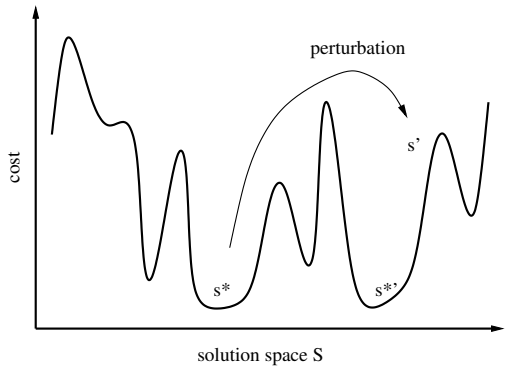

Figure 2: Pictorial representation of iterated local search. Starting with a local minimum s^*, we apply a perturbation leading to a less good solution s'. After applying LocalSearch, we find a new local minimum $s^{*\prime}$.

of iteration?". Our answer is "YES", and in fact the improvements obtained in practice are usually significant. Only in rather pathological cases where the iteration method is "incompatible" with the local search will the improvement be minimal. In the same vein, in order to have the *most* improvement possible, it may be necessary to have some understanding of the way the LocalSearch works. For the moment being, we wish to focus solely on the high-level architecture of iterated local search.

Let $\mathcal{C}$ be the cost function of our combinatorial optimization problem; $\mathcal{C}$ is to be *minimized*. We label candidate solutions or simply "solutions" by s, and denote by $\mathcal{S}$ the set of all s. Finally, the local search procedure LocalSearch defines a mapping from the set $\mathcal{S}$ to the smaller set $\mathcal{S}^*$ of locally optimal solutions s^*. Take an s or an s^* at random. Typically, the distribution of costs found has a very rapidly rising part at the lowest values. In Figure 1 we show the kind of distributions found in practice for combinatorial optimization problems having a finite solution space. The distribution of costs is bell-shaped, with a mean and variance that is significantly smaller for solutions in $\mathcal{S}^*$ than for those in $\mathcal{S}$. As a consequence, it is much better to sample in $\mathcal{S}^*$ than to sample in $\mathcal{S}$ if one seeks low cost solutions. Now the question is *how to go beyond this use of* LocalSearch*?*. More precisely, given the mapping from $\mathcal{S}$ to $\mathcal{S}^*$, how can one further reduce the costs found without opening up and modifying LocalSearch, leaving it as a "black box" routine?

Random restart The simplest possibility to improve upon a cost found by LocalSearch is to repeat the search from another starting point. Although such a "random restart" approach with independent samplings is sometimes a useful strategy (in particular if everything else fails), it breaks down with growing instance size because in that limit the tail of the distribution of costs collapses. Indeed, empirical studies [8] and general arguments [15] indicate that local search algorithms on large generic instances lead to costs that: (i) have a mean that is a fixed percentage excess above the optimum cost; (ii) have a *distribution* that becomes arbitrarily peaked about the mean when the instance size goes to infinity. This second property makes it impossible in practice to find an s^* whose cost is even a little bit lower than the typical cost. To reach very low cost configurations, a biased sampling is necessary; this is precisely what is accomplished by a stochastic search.

Iterated Local Search ILS tries to avoid the disadvantages of random restart by exploring $\mathcal{S}^*$ using a walk that steps from one s^* to a "nearby" one. This walk is heuristically desribed as follows. Given the current s^*, we first apply a change or perturbation that leads to an intermediate state s' (which belongs to $\mathcal{S}$). Then LocalSearch is applied to s' and we reach a solution $s^{*\prime}$ in $\mathcal{S}^*$. If $s^{*\prime}$ passes an acceptance test, it becomes the next element of the walk in $\mathcal{S}^*$; otherwise, one returns to s^*. The

resulting walk is a case of a stochastic search in $\mathcal{S}^*$. The working principle of the ILS procedure is illustrated in Figure 2.

In addition, the perturbations may depend on any of the previous s^*. In this case one has a walk in $\mathcal{S}^*$ with *memory*. Now the reader may have noticed that aside from the issue of perturbations (which use the structure on $\mathcal{S}$), our formalism reduces the problem to that of a stochastic search on $\mathcal{S}^*$. Then all of the bells and whistles (diversification, intensification, adaptive perturbations and acceptance criteria, etc...) that are used with other metaheuristics may be applied here. This leads us to define iterated local search algorithms as metaheuristics having the following high level architecture:

procedure *Iterated Local Search*
  s_0 = GenerateInitialSolution
  s^* = LocalSearch(s_0)
  repeat
    s' = Perturbation(s^*, *history*)
    $s^{*\prime}$ = LocalSearch(s')
    s^* = AcceptanceCriterion(s^*, $s^{*\prime}$, *history*)
  until termination condition met
end

In practice, much of the potential complexity of ILS is hidden in the history dependence. If there happens to be no such dependence, the walk has no memory: the perturbation and acceptance criterion do not depend on any of the solutions visited previously during the walk, and one accepts or not $s^{*\prime}$ with a fixed rule. This leads to random walk dynamics on $\mathcal{S}^*$ that are "Markovian", the probability of making a particular step from s_1^* to s_2^* depending only on s_1^* and s_2^*. Most of the work using ILS has been of this type, though recent studies show unambiguously that incorporating memory enhances performance [17].

3 Implementing Iterated Local Search

To implement an ILS algorithm, there are four components to consider: GenerateInitialSolution, LocalSearch, Perturbation, and AcceptanceCriterion. Although the final goal may be to develop a state-of-the-art algorithm, it is relatively straight-forward to first develop a more basic version of ILS. Indeed, (i) one can start with a random solution; (ii) for most problems a local search algorithm is readily available; (iii) for the perturbation, a random move in a neighborhood of higher order than the one used by the local search algorithm can be surprisingly effective; and (iv) a reasonable first guess for the acceptance criterion is to force the cost to decrease, corresponding to a first-improvement descent in the set $\mathcal{S}^*$. Basic ILS implementations of this type usually lead to much better performance than random restart approaches. The developer can then run this basic ILS to build his intuition and try to improve the overall algorithm performance by improving each of the four modules and tuning their interaction.

Which are the issues we have to consider for the optimization of the single components and their interaction? Let us illustrate some of these points using the example ILS application to the well known traveling salesman problem (TSP), which is probably also the most developped ILS application.

Initial solution. The local search application to s_0 defines the starting point s_0^* of the walk in $\mathcal{S}^*$. If very high quality solutions are to be reached as early as possible in the search, then starting from a best possible s_0^* becomes an important issue. Often, this is achieved by using greedy starting solutions. Yet, starting a local search from the best possible greedy solution does not necessarily result in the best possible s_0^*. We refer to [8] for some evidence of this fact using the TSP example application. For long runs, the influence of the initial solutions should become less important; if

on the contrary the influence of the initial solution persists, this may indicate that the ILS walk shows difficulties in exploring $\mathcal{S}^*$, which may be due to a poor choice of the perturbation or the acceptance criterion.

Perturbation. The goal here is to escape from local optima by applying perturbations to the current local minimum. An important feature is how strong the perturbations should be. If they are too small, one will often fall back to s^* and few new solutions of $\mathcal{S}^*$ will be explored. If on the contrary the perturbations are too large, s' will be almost random, there will be no bias in the sampling, and we will recover a random restart type algorithm. Ideally, the type of perturbation introduced into a solution should not be directly undoable by the local search and should complement it in some ways. For the TSP, a simple but effective perturbation is the so called *double-bridge move* which cuts the current tour at four positions and uses a particular way of reconnecting the four remaining tour segments. While, for the TSP a "good" perturbation is rather small, things may be different for other problems. An example is the quadratic assignment problem for which experimentally a relatively large perturbation size seems to be necessary [10]. Other important issues are the use of adaptive perturbations: *a priori* one may have no guess for what is the best perturbation strength or it may be instance dependent. The use of complex perturbations [5] or even the use of exact algorithms solving subparts of a problem to optimality [11] are active topics of research.

Finally, it should be mentioned that the use of small perturbations can lead to large speed-ups; the reason is that for the same computation time a much larger number of local searches can be applied than when starting from random initial solutions and typically this speed advantage increases with problem size.

Acceptance Criterion. The perturbation mechanism together with the local search defines the possible transitions between a current solution s^* in $\mathcal{S}^*$ to a "neighboring" solution $s^{*\prime}$ also in $\mathcal{S}^*$. The procedure AcceptanceCriterion then determines whether $s^{*\prime}$ is accepted or not as the new current solution. AcceptanceCriterion has a strong influence on the nature and effectiveness of the walk in $\mathcal{S}^*$. Roughly, it can be used, together with Perturbation, to control the balance between intensification and diversification of that search. For instance, a first improvement type descent in $\mathcal{S}^*$ is implemented by an acceptance criterion which only accepts better solutions. At the opposite extreme one can always accept the new solution irrespective of its cost; this leads to a random walk dynamics in $\mathcal{S}^*$. Many intermediate choices between these two extreme cases are possible, and in particular rather complex acceptance criteria that involve limited amount of directed diversification or intensification are possible [18].

Local search. Since the behavior and performance of the over-all ILS algorithm is quite sensitive to the choice of the embedded heuristic, one should optimize this choice whenever possible. In practice, there may be many quite different algorithms that can be used for the embedded heuristic. One might think that the better the local search, the better the corresponding ILS. Often this is true. For instance in the context of the TSP, Lin-Kernighan [9] is a better local search than 3-opt which itself is better than 2-opt [8]. Using a fixed type of perturbation such as the double-bridge move, one finds that iterated Lin-Kernighan gives better solutions than iterated 3-opt which itself gives better solutions than iterated 2-opt [8, 18]. But if we assume that the total computation time is fixed, it might be better to apply more frequently a faster but less effective local search algorithm than a slower and more powerful one.

Finally, it should be clear that the local search algorithm need not be a simple iterative improvement algorithms, but that any local search based on metaheuristic approaches like tabu search or simulated annealing may be used instead and often better performance may result. This is indeed the case in the job-shop scheduling problem: the use of tabu search as the embedded heuristic gives rise to a very effective iterated local search [12].

Global optimization of ILS. The global optimization of ILS concerns the tuning of the interactions between the individual components. While this is certainly a rather complex problem, here we can only give some rough guidelines. In our experience, the main dependencies of the components which have to be considered are:

1. the perturbation should not be easily undone by the local search; if the local search has obvious short-comings, a good perturbation should compensate for them.
2. the combination Perturbation–AcceptanceCriterion determines the relative balance of intensification and diversification and should receive an particularly strong attention. Large perturbations are only useful if they can be accepted, and that occurs only if the acceptance criterion is not too biased towards better solutions.

As a general guideline, LocalSearch should be as powerful as possible as long as it is not too costly in CPU time. Then, given a choice for that module, find a well adapted perturbation; to the extent possible, it should take advantage of the structure of the problem. Finally, set the AcceptanceCriterion routine so that $\mathcal{S}^*$ is sampled adequately. With such a recipe, the construction of an ILS is a bottom-up process. However, to improve the ILS, it is natural to then optimize each module assuming the others are fixed; this is a "local optimization" approach to the global optimization problem. When performing such optimizations, the interactions between the modules are essential, and for instance the balance between intensification and diversification is very important and remains a challenging problem.

4 Successful applications of ILS

ILS algorithms have been applied successfully to a variety of combinatorial optimization problems. In some cases, these algorithms achieve extremely high performance and even constitute the current state-of-the-art metaheuristics. The best example are the existing ILS approaches to the TSP, where ILS algorithms actually define the state-of-the-art in solving that problem. Another success story of ILS are applications to scheduling problems. In fact, for a number of scheduling problems the currently best known algorithms follow the ILS metaheuristic. Examples are the iterated Dynasearch algorithm by Congram, Potts and van de Velde [6], flow-shop scheduling type problems [16, 19], and job-shop scheduling problems [12, 2]. There exist a number of other applications of the ILS metaheuristic for which excellent results have been reported. For a detailed review of existing applications we refer to a longer overview article [10].

5 Conclusions

ILS has many of the most desirable features of a metaheuristic: it is simple, easy to implement, robust, and highly effective. The essential idea of ILS lies in focusing the search not on the full space of solutions but on a smaller subspace defined by the solutions that are locally optimal for a given optimization engine. The power of ILS lies in its *biased* sampling of the set of local optima. The efficiency of this sampling depends both on the kinds of perturbations and on the acceptance criteria. Interestingly, even with the most naïve implementations of these parts, ILS is much better than random restart. But with further work so that the different modules are well adapted to the problem at hand, ILS can often become a competitive or even state of the art algorithm. Such a dichotomy is important because the optimization of the algorithm can be done progressively, and so ILS can be kept at any desired level of simplicity. This, plus the modular nature of iterated local search, leads to short development times and gives ILS an edge over more complex metaheuristics in the world of industrial applications.

References

[1] D. Applegate, W. Cook, and A. Rohe. Chained Lin-Kernighan for large traveling salesman problems. Technical Report No. 99887, Forschungsinstitut für Diskrete Mathematik, University of Bonn, Germany, 1999.

[2] E. Balas and A. Vazacopoulos. Guided local search with shifting bottleneck for job shop scheduling. *Management Science*, 44(2):262–275, 1998.

[3] E. B. Baum. Towards practical "neural" computation for combinatorial optimization problems. In J. Denker, editor, *Neural Networks for Computing*, pages 53–64, 1986. AIP conference proceedings.

[4] J. Baxter. Local optima avoidance in depot location. *Journal of the Operational Research Society*, 32:815–819, 1981.

[5] B. Codenotti, G. Manzini, L. Margara, and G. Resta. Perturbation: An efficient technique for the solution of very large instances of the Euclidean TSP. *INFORMS Journal on Computing*, 8:125–133, 1996.

[6] R. K. Congram, C. N. Potts, and S. L. Van de Velde. An iterated dynasearch algorithm for the single–machine total weighted tardiness scheduling problem. INFORMS Journal on Computing, to appear, 2000.

[7] D. S. Johnson. Local optimization and the travelling salesman problem. In *Proceedings of the 17th Colloquium on Automata, Languages, and Programming*, volume 443 of *LNCS*, pages 446–461. Springer Verlag, Berlin, 1990.

[8] D. S. Johnson and L. A. McGeoch. The travelling salesman problem: A case study in local optimization. In E.H.L. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley & Sons, Chichester, England, 1997.

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

[10] H. R. Lourenço, O. Martin, and T. Stützle. Iterated local search. In F. Glover and G. Kochenberger, editors, *State-of-Art Handbook on Metaheuristics*. Kluwer Academic Publishers, To appear. A preliminary version is available at `http://www.intellektik.informatik.tu-darmstadt.de/tom/pub.html`.

[11] H. R. Lourenço. Job-shop scheduling: Computational study of local search and large-step optimization methods. *European Journal of Operational Research*, 83:347–364, 1995.

[12] H. R. Lourenço and M. Zwijnenburg. Combining the large-step optimization with tabu-search: Application to the job-shop scheduling problem. In I.H. Osman and J.P. Kelly, editors, *Meta-Heuristics: Theory & Applications*, pages 219–236. Kluwer Academic Publishers, 1996.

[13] O. Martin and S. W. Otto. Combining simulated annealing with local search heuristics. *Annals of Operations Research*, 63:57–75, 1996.

[14] O. Martin, S. W. Otto, and E. W. Felten. Large-step Markov chains for the traveling salesman problem. *Complex Systems*, 5(3):299–326, 1991.

[15] G. R. Schreiber and O. C. Martin. Cut size statistics of graph bisection heuristics. *SIAM Journal on Optimization*, 10(1):231–251, 1999.

[16] T. Stützle. Applying iterated local search to the permutation flow shop problem. Technical Report AIDA–98–04, FG Intellektik, TU Darmstadt, August 1998.

[17] T. Stützle. *Local Search Algorithms for Combinatorial Problems — Analysis, Improvements, and New Applications*. PhD thesis, Darmstadt University of Technology, Department of Computer Science, 1998.

[18] T. Stützle and H. H. Hoos. Analyzing the run-time behaviour of iterated local search for the TSP. In P. Hansen and C. Ribeiro, editors, *Essays and Surveys in Metaheuristics*. Kluwer Academic Publishers, 2001. To appear.

[19] Y. Yang, S. Kreipl, and M. Pinedo. Heuristics for minimizing total weighted tardiness in flexible flow shops. *Journal of Scheduling*, 3(2):89–108, 2000.