A Gentle Introduction to Iterated Local Search*

Helena Ramalhinho Lourenço * Olivier Martin † Thomas Stützle ‡

* Universitat Pompeu Fabra, Barcelona, Spain
FULL ADDRESS TO BE ADDED
Email: helena.ramalhin@econ.upf.es

† Université Paris-Sud, Orsay, France
FULL ADDRESS TO BE ADDED
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 cases the practically 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. If we think of a metaheuristic as simply a construction for guiding (problem-specific) heuristics, the ideal case is when the metaheuristic can be used without *any* problem-dependent knowledge.

As metaheuristics have become more and more sophisticated, this ideal case has been pushed aside in the quest for greater performance. As a consequence, problem-specific knowledge (in addition to that built into the heuristic being guided) must now be incorporated into metaheuristics in order to reach the state of the art level. Unfortunately, this makes the boundary between heuristics and *meta*heuristics fuzzy, and we run the risk of loosing both simplicity and generality. To counter this, we appeal to modularity and try to decompose a metaheuristic algorithm into a few parts, each with their own specificity. In particular, we would like to have a totally general purpose part, while problem-specific knowledge built into the metaheuristic would be restricted to an embedded heuristic. Finally, to the extent possible, we prefer to leave untouched the embedded heuristic (which is to be "guided") because of its potential complexity. One can also consider the case where this heuristic is only available through an object module, the source code being proprietary; it is then necessary to be able to treat it as a "black-box" routine. Iterated local search provides a simple way to satisfy all these requirements.

The essence of the iterated local search (ILS) metaheuristic can be given in a nut-shell: one *iteratively* builds a sequence of solutions generated by the embedded heuristic, leading to far better solutions than if one were to use repeated random trials of that heuristic. This simple idea [7] has a long history, and its rediscovery by many authors has lead to many different names for ILS like *iterated descent* [6, 5], *large-step Markov chains* [18], *iterated Lin-Kernighan* [11], *chained local optimization* [17], or combinations of these [2] ... Readers interested in these historical developments should consult the review [12]. For us, there are two main points that make an algorithm an ILS: (i) there is a single chain that is followed;

*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.

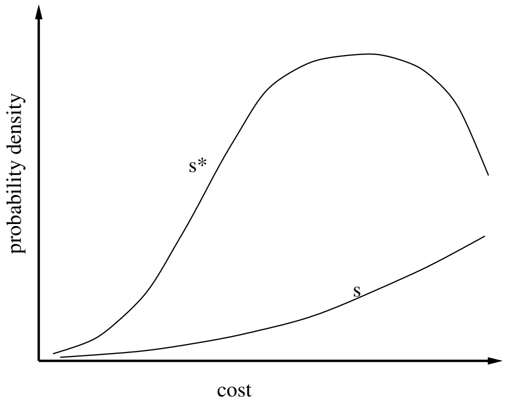

Figure 1: Probability densities of costs. The curve labeled s gives the cost density for all solutions, while the curve labeled s^* gives it for the solutions that are local optima.

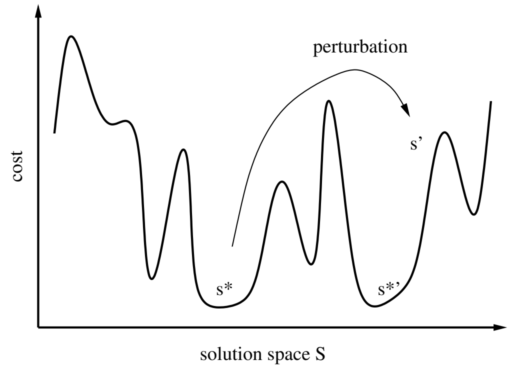

Figure 2: Pictorial representation of iterated local search. Starting with a local minimum s^*, we apply a perturbation leading to a less good solution s'. After applying LocalSearch, we find a new local minimum $s^{*\prime}$.

(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.

The purpose of this review is to give a description of the underlying principles of ILS and to show where it stands in terms of performance. So far, in spite of its conceptual simplicity, it has lead to a number of state-of-the-art results without the use of too much problem-specific knowledge; perhaps this is because iterated local search is very malleable, many implementation choices being left to the developer.

2 Iterating a local search

General framework: We assume we have been given a problem-specific approximate algorithm that from now on we shall refer to as a local search (even if in fact it is not a true local search). This algorithm is implemented via a computer routine that we call LocalSearch. The question we ask is "Can such an algorithm be improved by the use 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 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 (for simplicity $\mathcal{S}$ is taken to be finite, but it does not matter much). 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 randomly 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. Every s^* generated is then independent, and the use of multiple trials allows one to reach into the lower part of the distribution. Although such a "random restart" approach with independent samplings is sometimes a useful strategy (in particular when all other options fail), it breaks down as the instance size grows because in that limit the tail of the distribution of costs collapses. Indeed, empirical studies [12] and general arguments [19] 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. Note however that there do exist many solutions of significantly lower cost, it is just that *random* sampling has a lower and lower probability of finding them as the instance size increases. To reach those 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 $\mathcal{S}^*$. This iterated local search procedure should lead to good biased sampling as long as the perturbations are neither too small nor too large. 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 random, there will be no bias in the sampling, and we will recover a random restart type algorithm.

The overall ILS procedure is pictorially illustrated in Figure 2. To be complete, let us note that generally the iterated local search walk will not be reversible; in particular one may sometimes be able to step from s_1^* to s_2^* but not from s_2^* to s_1^*. However this "unfortunate" aspect of the procedure does not prevent ILS from being very effective in practice.

Since deterministic perturbations may lead to short cycles, one should randomize the perturbations or have them be adaptive so as to avoid this kind of cycling. If the perturbations depend on any of the previous s^*, 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, tabu, adaptive perturbations and acceptance criteria, etc...) that are commonly used in the contexts of 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 [21].

3 Implementing Iterated Local Search

To implement anILS algorithm, there are four components to consider: GenerateInitialSolution, LocalSearch, Perturbation, and AcceptanceCriterion. Before attempting to develop a state-of-the-art algorithm, it is relatively straight-forward to develop a more basic version of ILS. Indeed, (i) one can start with a random solution or one returned by some greedy construction heuristic; (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.

This should be particularly effective if it is possible to take into account the specificities of the combinatorial optimization problem under consideration. In practice, this tuning is easier for ILS than for memetic algorithms or tabu search to name but these metaheuristics. The reason may be that the complexity of ILS is reduced by its modularity, the function of each component being relatively easy to understand. Finally, the last task to consider is the overall optimization of the ILS algorithm; indeed, the different components affect one another and so it is necessary to understand their interactions.

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, while in other cases the ILS approach is merely competitive with other metaheuristics.

The most prominent ILS application is certainly that to the travelling salesman problem (TSP). Probably the oldest attempt is due to Baum [6, 5]. He coined his method *iterated descent*; his tests used 2-opt as the embedded heuristic, random 3-changes as the perturbations, and imposed the tour length to decrease (thus the name of the method). His results were not impressive, in part because he considered the non-Euclidean TSP, which in practice is substantially more difficult than the Euclidean TSP. A major improvement in the performance of ILS algorithms came from the *large-step Markov chain* (LSMC) algorithm proposed by Martin, Otto, and Felten [18]. They used a simulated annealing like acceptance criterion (LSMC) from which the algorithm's name is derived and considered both the application of 3-opt local search and the Lin-Kernighan heuristic (LK) which is the best performing local search algorithm for the TSP. But probably the key ingredient of their work is the introduction of the double-bridge move for the perturbation. This choice made the approach very powerful for the Euclidean TSP, and that encouraged much more work along these lines. In particular, Johnson [11, 12] coined the term "iterated Lin-Kernighan" (ILK) for his implementation of ILS using the Lin-Kernighan as the local search. We refer to Johnson and McGeoch [12] for a summary of the situation as of 1997. Since then a number of additional ILS variants for the TSP were developed [13, 21, 22] and currently the highest performance ILS for the TSP is the chained LK code by Applegate, Bixby, Chvatal, and Cook [1] which is available as a part of the Concorde software package at www.keck.caam.rice.edu/-concorde.html. Furthermore, Applegate, Cook, and Rohe [2] performed thorough experimental tests of this code.

A second, major application field, where ILS algorithms have shown to be very competitive, is scheduling. In fact, for a number of scheduling problems th currently best known algorithms follow the ILS metaheuristic. Examples are the iterated Dynasearch algorithm by Congram, Potts and van de Velde [10] for the Single Machine Total Weighted Tardiness Problem (SMTWTP), several single and parallel machine problems attacked in [8, 9], flow-shop scheduling type problems [20, 23], and job-shop scheduling problems [14, 15, 16, 3].

4.0.1 MAX-SAT

Battiti and Protasi present an application of *reactive search* to the MAX-SAT problem [4]. Their algorithm consists of two phases: a local search phase and a diversification (perturbation) phase. Because of this, their approach fits perfectly into the ILS framework. Their perturbation is obtained by running a tabu search on the current local minimum so as to guarantee that the modified solution s' is sufficiently different from the current solution s^*. Their measure of difference is just the Hamming distance; the minimum distance is set by the length of a tabu list that is adjusted during the run of the algorithm. For the LocalSearch, they use a standard greedy descent local search for the MAX-SAT problem. Depending on the distance between $s^{*\prime}$ and s^*, the tabu list length for the perturbation phase is dynamically adjusted. The next perturbation phase is then started based on solution $s^{*\prime}$—corresponding to the `RW` acceptance criterion. This work illustrates very nicely how one can adjust dynamically the perturbation strength in an ILS run. We conjecture that similar schemes will prove useful to optimize ILS algorithms in a nearly automatic way.

5 Conclusions

We can summarize this section by saying that the potential power of iterated local search 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, iterated local search is much better than random restart. But still much better results can be obtained if the iterated local search modules are optimized. First, the acceptance criteria can be adjusted empirically as in simulated annealing without knowing anything about the problem being optimized. This kind of optimization will be familiar to any user of metaheuristics, though the questions of memory may become quite complex. Second, the Perturbation routine can incorporate as much problem-specific information as the developer is willing to put into it. In practice, a rule of thumb can be used as a guide: "a good perturbation transforms one excellent solution into an excellent starting point for a local search. Together, these different aspects show that iterated local search algorithms can have a wide range of complexity, but complexity may be added progressively and in a modular way. (Recall in particular that all of the fine-tuning that resides in the embedded local search can be ignored if one wants, and it does not appear in the metaheuristic per-se.) This makes iterated local search an appealing metaheuristic for both academic and industrial applications. The cherry on the cake is speed: as we shall soon see, one can perform k local searches embedded within an iterated local search *much* faster than if the k local searches are run within random restart.

ILS has many of the 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 success of ILS lies in the *biased* sampling of this set of local optima. How effective this approach turns out to be depends mainly on the choice of the local search, the perturbations, and the acceptance criterion. Interestingly, even when using the most naïve implementations of these parts, ILS can do 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.

This 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.

The ideas and results presented in this chapter leave many questions unanswered. Clearly, more work needs to be done to better understand the interplay between the ILS modules GenerateInitialSolution, Perturbation, LocalSearch, and Perturbation. In particular, we expect significant improvements to arise through the intelligent use of memory, explicit intensification and diversification strategies, and greater problem-specific tuning. The exploration of these issues has barely begun but should lead to higher performance iterated local search algorithms.

References

[1] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Finding tours in the TSP. Preliminary version of a book chapter available via www.keck.caam.rice.edu/concorde.html, 2000.

[2] 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.

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

[4] R. Battiti and M. Protasi. Reactive search, a history-based heuristic for MAX-SAT. *ACM Journal of Experimental Algorithmics*, 2, 1997.

[5] E. B. Baum. Iterated descent: A better algorithm for local search in combinatorial optimization problems. Technical report, Caltech, Pasadena, CA, 1986. manuscript.

[6] 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.

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

[8] P. Brucker, J. Hurink, and F. Werner. Improving local search heuristics for some scheduling problems — part I. *Discrete Applied Mathematics*, 65(1–3):97–122, 1996.

[9] P. Brucker, J. Hurink, and F. Werner. Improving local search heuristics for some scheduling problems — part II. *Discrete Applied Mathematics*, 72(1–2):47–69, 1997.

[10] 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.

[11] 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.

[12] 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.

[13] K. Katayama and H. Narihisa. Iterated local search approach using genetic transformation to the traveling salesman problem. In *Proc. of GECCO'99*, volume 1, pages 321–328. Morgan Kaufmann, 1999.

[14] S. Kreipl. A large step random walk for minimizing total weighted tardiness in a job shop. *Journal of Scheduling*, 3(3):125–138, 2000.

[15] 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.

[16] 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.

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

[18] 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.

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

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

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

[22] T. Stützle and H. H. Hoos. Analyzing the run-time behaviour of iterated local search for the TSP. Technical Report IRIDIA/2000-01, IRIDIA, Université Libre de Bruxelles, 2000. Available at http://www.intellektik.informatik.tu-darmstadt.de/˜tom/pub.html.

[23] 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.