Population-Based Iterated Local Search: Restricting Neighborhood Search by Crossover

Dirk Thierens

Institute of Information and Computing Sciences
Utrecht University, The Netherlands
dirk.thierens@cs.uu.nl

Abstract. Iterated local search (ILS) is a powerful meta-heuristic algorithm applied to a large variety of combinatorial optimization problems. Contrary to evolutionary algorithms (EAs) ILS focuses only on a single solution during its search. EAs have shown however that there can be a substantial gain in search quality when exploiting the information present in a population of solutions. In this paper we propose the use of a population for ILS. We define the general form of the resulting meta-heuristic, called population-based iterated local search (PILS). PILS is a minimal extension of ILS that uses previously generated solutions in the neighborhood of the current solution to restrict the neighborhood search by ILS. This neighborhood restriction is analogous to the way crossover preserves common substructures between parents when generating offspring. To keep the discussion concrete, we discuss a specific instantiation of the PILS algorithm on a binary trap function.

1 Introduction

In the field of meta-heuristics a large - and increasing - set of algorithms is studied to tackle hard combinatorial optimization problems typically found in Artificial Intelligence and Operational Research [11][13][16]. Most meta-heuristics try to find better solutions by exploring the neighborhood of a single solution. Examples are tabu search, simulated annealing, variable neighborhood search, greedy adaptive search procedure, and iterated local search. Other meta-heuristics bias their search by exploiting the information contained in a multi-set of current solutions. Examples of these population-based methods are evolutionary algorithms, ant colony optimization, and scatter search. All these meta-heuristics have their own view on how an efficient search process might be build. However, it is important to realize that their underlying philosophies are not necessarily incompatible. This observation creates the opportunity to design new meta-heuristics by combining the driving mechanism of two (or more) meta-heuristics. In this paper we propose a meta-heuristic that combines the power of iterated local search with the principle of extracting useful information about the search space by keeping a population of solutions. We call the resulting algorithm population-based iterated local search (PILS). As in ILS, PILS tries

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 234–245, 2004.

to improve on a single solution by running an iterated local search (ILS) algorithm. In addition to ILS, PILS also keeps a small population of neighboring solutions. These solutions are used to focus the iterated local search process to the common subspace between the current solution and one of the population members. The underlying assumption of PILS is that neighboring local optima share common substructures that could be exploited to generate new solutions. More specifically, PILS restricts the perturbation of ILS to the subspace where the current solution and a population member disagree, thus preserving their common substructure.

In the next section we formally describe the PILS meta-heuristic. This general algorithmic framework has to be instantiated for a particular problem type. In Section 3 we discuss such an instantiation for the trap function. Section 4 discusses related and future work. Finally, Section 5 concludes this preliminary work on the PILS meta-heuristic.

2 Population-Based Iterated Local Search

PILS is a population-based extension of the ILS meta-heuristic. ILS applies a local search procedure to explore the neighborhood of the current solution in search for a local optimum [8]. When a local optimum is reached, ILS perturbs this solution and continues the local search from this new solution. This perturbation should be large enough such that the local search does not return to the same local optimum in the next iteration. However the perturbation should not be too large, otherwise the search characteristics will resemble those of a multi-start local search algorithm. As long as the termination condition has not been met ILS continues its search this way. The underlying design philosophy of ILS - or in other words, its inductive bias - is to perform a stochastic neighborhood search in the space of local optima. Naturally, we do not have an operator that directly generates the neighborhood in this space, but the combination of the perturbation operator and the local search operator, both working in the original solution space, achieves this in an indirect way. The algorithmic description of ILS is as follows:

```
ITERATED LOCAL SEARCH()
1  S ← GENERATEINITIALSOLUTION
2  S ← LOCALSEARCH(S)
3  while NOTTERMINATED?(S)
4     do S' ← PERTURBATION(S, history)
5        S' ← LOCALSEARCH(S')
6        S ← ACCEPTANCECRITERION(S, S', history)
7  return S
```

The goal of the population-based iterated local search meta-heuristic is to improve upon the ILS algorithm by keeping a population of neighboring solutions, and exploit this information to focus the ILS algorithm. PILS explores the neighborhood of the current solution in two ways. With probability *Pratio*

it simply applies the regular ILS procedure. In the other case, a random member of the population is chosen and the `Perturbation`, and if meaningful for the particular problem, the `LocalSearch`, are restricted to the search subspace where the current solution and the population member disagree. The general algorithmic description of PILS is as follows:

```
Population-based Iterated Local Search()
 1  Pop ← CreateInitialPop(PopSize)
 2  Pop[0] ← AcceptanceCriterion(Pop)
 3  while NotTerminated?(Pop)
 4     do if CoinFlip(Pratio)
 5           then S' ← Perturbation(Pop[0], history)
 6                S' ← LocalSearch(S')
 7                Pop ← AcceptanceCriterion(Pop, S', history)
 8           else i ← RandomInt(1, PopSize)
 9                S' ← Perturbation(Pop[0], Pop[i], history)
10                S' ← LocalSearch(S', Pop[i])
11                Pop ← AcceptanceCriterion(Pop, S', history)
12  return Pop[0]
```

The acceptance criterion could be the requirement that new solutions should have a better (or at least equal) fitness value than the current solution. Another criterion - applied in simulated annealing - accepts new solutions with a probability depending on the fitness difference between the current and new solution. The acceptance criterion might also keep a history list such that its decision depends on previously visited solutions. This approach is the main mechanism of tabu search.

When applying PILS to a particular class of problems the algorithmic framework needs to be instantiated in accordance with the characteristics of the underlying problem. To illustrate this we have applied the PILS meta-heuristic to a trap function, which is a hard and well studied optimization problem in EA research [1][3][15]. In the next section we specify one possible instantiation of the PILS meta-heuristic for the trap functions.

3 Example

3.1 Instantiating PILS for Trap Functions

In the previous section we have described the general population-based iterated local search algorithm. A specific instantiation of PILS requires the specification of the procedures `GenerateInitialSolution` (or `CreateInitialPop`), `LocalSearch`, `NotTerminated?`, `Perturbation`, and `AcceptanceCriterion`. In the remainder of this paper we will look at a specific PILS algorithm applied to a hard optimization problem in a fixed-length binary search space. First, we specify the ILS algorithm for this problem. `GenerateInitialSolution` simply generates an initial fixed-length binary string at random. The `NotTerminated?` test ensures

that we continue searching until we have reached the global maximum or we have exceeded a maximum number of trials. The `Perturbation` operator flips the bits with a fixed probability. The `LocalSearch` and the `AcceptanceCriterion` are combined to create a first-improvement hill-climbing algorithm. Each bit in the current string is flipped individually and the resulting fitness value is computed. Whenever there is a fitness improvement, or when the new value equals the previous one, the bit change is accepted. The search continues flipping bits at the current position in the string.

The PILS algorithm extends the ILS by exploiting information stored in other solutions in the population. More specifically, we adapt the `LocalSearch` and the `Perturbation` mechanisms by restricting the bits that they are allowed to change. `Perturbation` and `LocalSearch` now take two solutions as input: one is the current solution the search is focusing on, and the other is a randomly chosen solution from the population, which acts as a mask. The `Perturbation` operator flips with fixed probability only those bits that have a different value in the current solution and the mask. The combined `LocalSearch` and `AcceptanceCriterion` also restrict their first-improvement hill-climbing search to those bits that have a different value in the current solution and the mask. When a new solution is accepted it replaces the current solution which in turn replaces the worst solution in the population. The algorithmic description of the PILS meta-heuristic for the binary problem is given by:

```
PERTURBATION(BitString, MaskBitString, ProbMut)
1  for i ← 1 to LENGTH(BitString)
2      do if BitString(i) ≠ MaskBitString(i)
3            then if COINFLIP(ProbMut)
4                    then BitString ← FLIPBIT(Bitstring, i)
5  return BitString

FIRSTIMPROVEMENTHILLCLIMBING(BitString, MaskBitString)
1  for i ← 1 to StringLength
2      do if BitString(i) ≠ MaskBitString(i)
3            then NewBitString ← FLIPBIT(Bitstring, i)
4                 if FITNESS(NewBitString) ≥ FITNESS(BitString)
5                    then BitString ← NewBitString
6  return BitString
```

The underlying assumption of the PILS algorithm is that neighboring local optima have some common substructures. By matching the current solution and a neighboring solution we try to identify these common substructures during the search process. The adaptive restriction of the ILS to a neighborhood with smaller dimension has two advantages. First, it ensures that the perturbation mechanism does not destroy the possibly useful common substructures. Second, the dimensionality reduction of the neighborhood to search might save considerable computational cost.

3.2 Fitness Function

To illustrate the use of the above PILS algorithm we measured its performance on a trap function consisting of misleading subfunctions of different lengths. Specifically, the fitness function $F(X)$ is constructed by adding subfunctions of length 1 (F_1), 2 (F_2), and 3 (F_3). Each subfunction has two optima: the optimal fitness value is obtained for an all-ones string, while the all-zeroes string represents a local optimum. The fitness of all other string in the subfunction is determined by the number of zeroes: the more zeroes the higher the fitness value. This causes a large basin of attraction toward the local optimum when applying the first-improvement hill-climber. The fitness values for the subfunctions are specified in Table 1 where the columns indicate the number of ones in the subfunctions F_1, F_2, and F_3. The fitness function $F(X)$ is composed of 4 subfunctions F_3, 6 subfunctions F_2, and 12 subfunctions F_1. The overall string-length of the problem is thus 36 (= 4 x 3 + 6 x 2 + 12 x 1). $F(X)$ has $2^{10} = 1024$ optima of which only one is the global optimum: the string with all ones having a fitness value of 220.

$$F(x_0 \dots x_{35}) = \sum_{i=0}^{3} F_3(x_{3i}x_{3i+1}x_{3i+2}) + \sum_{i=0}^{5} F_2(x_{12+2i}x_{12+2i+1}) + \sum_{i=0}^{11} F_1(x_{24+i})$$

Table 1. The fitness values of the subfunctions F_i of length i; the columns represent the number of bits in the subfunction that are equal to one.

	0	1	2	3
F_3	4	2	0	10
F_2	5	0	10	
F_1	0	10		

3.3 Analysis

It is shown in [10] that the most difficult part for a local search algorithm when optimizing trap functions is the last step where a single, largest suboptimal subfunction needs to be changed in the optimal substring while preserving all the other optimal subfunctions. It is instructive to compute the probability this will take place with ILS and the PILS. Call *Pmut* the probability that a single bit is flipped by the `Perturbation` procedure. If we assume - without loss of generality - that the first-improvement hill-climber (FIHC) visits all bits from left to right then the basins of attraction of the two optima are:

111	111 011
000	000 100 010 001 110 101

The probability $Pr[\mathbf{xxx} \rightarrow \mathbf{yyy}]$ that the substring **xxx** is changed into **yyy** by the perturbation and local search (FIHC) algorithm is given by:

$Pr[111 \to 111]$	$(1-p_{mut})^2$
$Pr[111 \to 000]$	$1-(1-p_{mut})^2$
$Pr[000 \to 000]$	$1-p_{mut}^2$
$Pr[000 \to 111]$	p_{mut}^2
$Pr[11 \to 11]$	$1-p_{mut}$
$Pr[11 \to 00]$	p_{mut}
$Pr[00 \to 00]$	$1-p_{mut}$
$Pr[00 \to 11]$	p_{mut}
$Pr[1 \to 1]$	1
$Pr[1 \to 0]$	0
$Pr[0 \to 0]$	0
$Pr[0 \to 1]$	1

The probability that the ILS algorithm generates the optimal string when only one order-3 subfunction is incorrect is given by the probability that changes the suboptimal substring, while preserving the other optimal substrings:

$$\begin{aligned} Pr(success) &= Pr[000 \to 111](Pr[111 \to 111])^3(Pr[11 \to 11])^6 \\ &= p_{mut}^2(1-p_{mut})^{12}. \end{aligned}$$

The PILS algorithm can only generate the optimal string from the current solution with one suboptimal subfunction if the chosen solution from the population has the optimal substring at the corresponding position. Assuming the chosen population member has only one other suboptimal subfunction the probability that the PILS algorithm now generates the optimal solution is given by:

$$\begin{aligned} Pr(success) &= Pr[000 \to 111]Pr[111 \to 111] \\ &= p_{mut}^2(1-p_{mut})^2. \end{aligned}$$

Naturally, the solution picked from the population will quite often not have the optimal subfunction at the required position but this will be compensated by the much higher probability of generating the global optimal string in case the condition is fulfilled. Figure 1 shows the difference between the probabilities of generating the optimal string for both ILS and PILS in the above circumstances.

3.4 Experimental Results

To see how PILS performs for the above trap function and to investigate its sensitivity to certain parameter choices, we have performed some experiments and measured the number of function evaluations needed to generate the optimal string for the first time. The parameters studied are the population size N, the perturbation strength P_{mut}, the perturbation strength in the reduced subspace $P_{maskmut}$, and the ratio P_{ratio} between the ILS and the dimensionality reduced ILS. All results are reported by plotting the experimental cumulative distribution function of the probability of generating the global optimal solution as a function

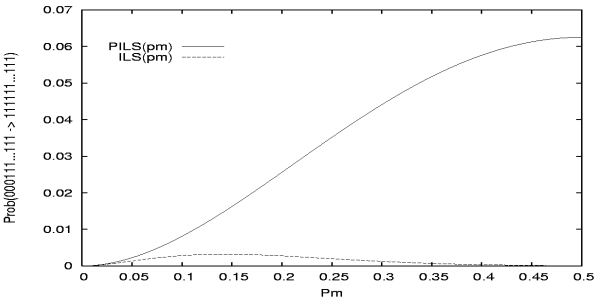

Fig. 1. Probability optima string generated from 000111...111 by ILS and PILS

of the number of function evaluations which is limited to a maximum of 100, 000. Every experimental CDF is calculated by running 1000 independent runs of the algorithm for a particular set of parameters.

1. Figure 2 shows the success probability as a function of the number of fitness function evaluation for ILS with varying perturbation strength. It can be observed that if the perturbation is too low the probability of finding a new local optimum is too low and the performance drops. If on the other hand the perturbation is too high the ILS becomes too destructive and its performance also degrades. Note that when $Pmut = 0.5$ ILS becomes a multi-start local search. From the size of the basins of attraction it is easy to calculate the success probability of generating the optimal string in this case, which is only 1 out of 2^{14} !

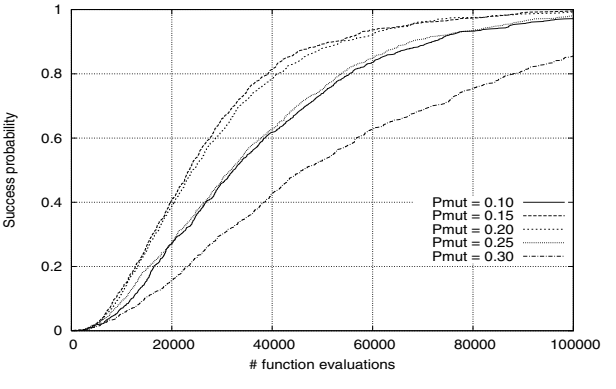

Fig. 2. ILS for varying perturbation strengths $Pmut$

2. In Figure 3 we have plotted the results of PILS for varying perturbation strengths when applying ILS to the entire - this is, the unrestricted - search space. The other parameters are fixed ($N = 5$, $Pratio = 0.5$, $Pmaskmut = 0.25$). Compared to the previous figure it is clear that PILS requires less function evaluations to reach a given performance level. In addition, PILS also seems to be more robust to the choice of parameter value of the perturbation strength.

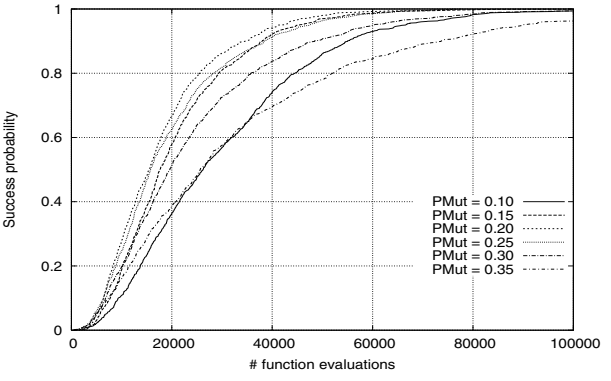

Fig. 3. PILS for varying perturbation strengths $Pmut$ of the unrestricted perturbation with $N = 5$, $Pratio = 0.5$, $Pmaskmut = 0.25$

3. The previous experiments showed the sensitivity of ILS (either as standalone or within PILS) for the perturbation strength $Pmut$. Figure 4 shows that PILS is rather insensitive to the perturbation strength in the reduced search space when applying the dimensionality restricted ILS. Even when all bits in the restricted subspace are fully randomized ($Pmaskmut = 0.5$), the performance stays good.
4. Figure 5 shows that PILS is also rather insensitive to the size of the population of which the neighboring solutions are chosen to compute the restricted subspace to explore.
5. Finally in Figure 6 we have plotted the results for different values of the probability of applying unrestricted ILS or the restricted ILS. $Pratio = 1.00$ is actually the standard ILS algorithm: clearly PILS improves on this in each case. Note that $Pratio = 0.00$ makes no sense because parts of the search space would become unaccessible to the search algorithm.

4 Discussion

PILS tries to improve the performance of ILS by keeping a population of neighboring solutions to reduce the dimensionality of the neighborhood to be explored

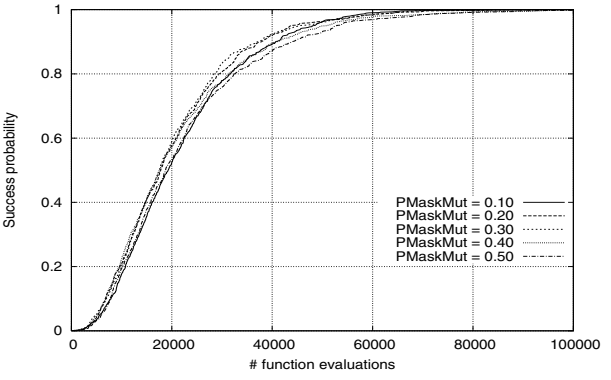

Fig. 4. PILS for varying perturbation strengths $Pmaskmut$ of the restricted perturbation with $N = 5$, $Pratio = 0.5$, $Pmut = 0.15$

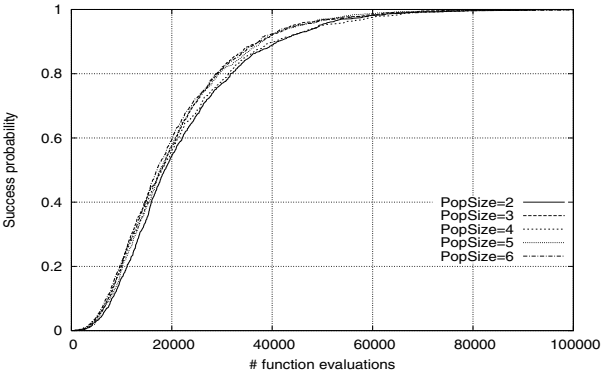

Fig. 5. PILS for varying population sizes N with $Pratio = 0.5$, $Pmut = 0.15$, $Pmaskmut = 0.25$

by ILS. By shielding off the common subspace between the current solution and a neighboring solution, PILS reduces the probability that common substructures found in good solutions are destroyed by the perturbation operator. Obviously, this mechanism is not unlike crossover in genetic algorithms. Although Holland [6] and Goldberg [4] consider crossover to be an operator that recombines or juxtaposes different substructures of two parent solutions, some crossover operators described in the literature could better be viewed as a perturbation operator in the search subspace defined by those variables where the two parent solutions have no common values. A good example in the binary search space is parameterized uniform crossover (PUX) where allele values between two strings are flipped with a fixed probability [14]. In fact because flipping equal bit values

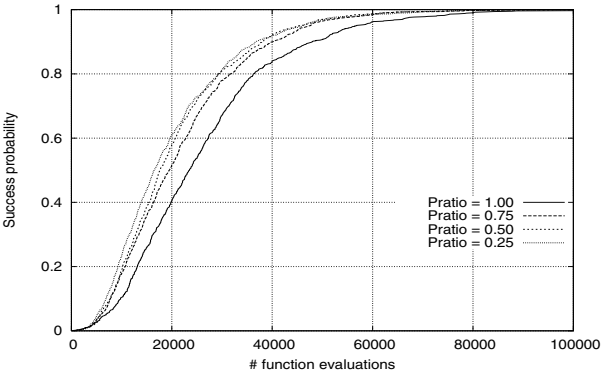

Fig. 6. PILS for varying probabilities *Pratio* between unrestricted and restricted perturbation and local search with $N = 5$, $Pmut = 0.15$, $Pmaskmut = 0.25$

does not produce a change, PUX is actually generating the same strings as the restricted perturbation operator we have applied in PILS.

The combination of local search with a population of solutions is also found in hybrid genetic algorithms, or memetic algorithms [2][9]. Memetic algorithms are basically genetic algorithms that search in the space of local optima. Therefore, the recombination operator will in general try to juxtapose different substructures from the two parents. Also, selection is applied to let all solutions in the population compete for their survival. When using selection the population should not be too small to avoid the premature loss of diversity which would lead to unsatisfying search results. In many combinatorial optimization tasks the local search operator is a computationally expensive procedure, and memetic algorithms often need to limit their use of the local search operator due to the high computational cost. In PILS the search is only focused on a single - for instance, the current best - solution as in ILS. No selection between members of a population is taking place, other than replacing the current solution by a new neighboring solution. The population in PILS are highly fit solutions encountered during the search, and are only used to help define a reduced neighborhood around the current solution.

Recently, it has been brought to our attention that Reeves also recognized the role crossover might play as a neighborhood restricting operator [12]. Reeves focused on the description of the working of a GA in terms of a generalized neighborhood search algorithm. He also noted that even the restricted neighborhood would in general be too large to search exhaustively, and while a traditional GA can be viewed as taking a random sample from that neighborhood, Reeves advocated the use of a more systematic local search. In PILS the restricted neighborhood is searched by ILS, which is a very efficient neighborhood search algorithm.

5 Conclusions

We have proposed the population-based iterative local search (PILS) algorithm. PILS aims to extend the efficiency of the iterative local search algorithm by exploiting information contained in a population of neighboring solutions. By using a population PILS restricts the ILS algorithm to explore only lower dimensional neighborhoods. The gain in efficiency is obtained at two levels. First, the perturbation operator is restricted to explore only the subspace where the current solution and a neighboring solution from the population disagree. This way promising, partial solutions remain untouched. Second, the local search operator only needs to search a neighborhood of smaller size. We have analyzed and tested PILS on trap functions, showing how and why PILS can be more efficient than ILS. A key assumption of the PILS algorithm is that local optimal solutions possess common substructures that can be exploited to increase the efficiency of the ILS. In future work we will investigate the use of PILS on other combinatorial optimization problems.

Acknowledgments. I would like to thank Ken De Jong, Colin Reeves, and Peter Merz for the valuable discussion at the Dagstuhl-seminar 04081 'Theory of Evolutionary Algorithms'.

References

1. D.H. Ackley. *A connectionist machine for genetic hill climbing*, Kluwer Academic Publishers, 1987.
2. D. Corne, F. Glover, and M. Dorigo (eds.). *New Ideas in Optimization*, McGraw-Hill, 1999.
3. K. Deb, and D.E. Goldberg. Analysing deception in trap functions. *Proceedings of Foundations of Genetic Algorithms*, pages 93–108, Morgan Kaufmann, 1993.
4. D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.
5. D. Frost, I. Rish, and L. Vila. Summarizing CSP hardness with continuous probability distributions. *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 327–334, AAAI-Press, 1997.
6. J.H. Holland. *Adaptation in Natural and Artificial Systems.* Michigan University Press, Ann Arbor, 1975.
7. H. Hoos, and T. Stützle. Towards a Characterisation of the Behaviour of Stochastic Local Search Algorithms for SAT. *Artificial Intelligence*, vol. 112, nrs. 1-2, pages 213-232, Elsevier Science Publishers, 1999.
8. H.R. Lourenço, O. Martin, and T. Stützle. A Beginner's Introduction to Iterated Local Search. *Proceedings of the 4th Metaheuristics International Conference*, 2001.
9. P. Merz, and B. Freisleben. Fitness Landscapes and Memetic Algorithm Design. *New Ideas in Optimisation*, D. Corne, M. Dorigo, and F. Glover (eds.) McGraw-Hill, 1999.
10. H. Mühlenbein. How genetic algorithms really work. I. Mutation and Hillclimbing. *Proceedings of Parallel Problem Solving from Nature*, pages 15–25, North-Holland, 1992.

11. I.H. Osman, and J.P. Kelly (eds.). *Meta-Heuristics: The Theory and Applications*, Kluwer Academic Publishers, Boston, 1996.
12. C.R. Reeves. Genetic Algorithms and Neighbourhood Search. *Proceedings of Evolutionary Computing, AISB Workshop*, pages 115–130, 1994.
13. C.C. Ribeiro, and P. Hansen (eds.). *Essays and Surveys in Metaheuristics*, Kluwer Academic Publishers, Boston, 2001.
14. W.M. Spears, and K.A. De Jong On the Virtues of Parametrized Uniform Crossover. *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230-236, Morgan Kaufmann, 1991.
15. D. Thierens, and D. Goldberg. Mixing in genetic algorithms. *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 38-45, Morgan Kaufmann, 1993.
16. S. Voss, S. Martello, I.H. Osman, and C. Roucairol (eds.). *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Kluwer Academic Publishers, Boston, 1999.