Available online at www.sciencedirect.com

SCIENCE DIRECT®

European Journal of Operational Research 151 (2003) 247–252

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/dsw

Preface

Guidelines for the use of meta-heuristics in combinatorial optimization

Alain Hertz [a,*], Marino Widmer [b]

[a] *École Polytechnique de Montréal – GERAD, 3000, Chemin de la Côte-Sainte-Catherine, Montréal (QC), Canada H3T 2A7*
[b] *Université de Fribourg – DIUF Faucigny 2, CH-1700 Fribourg, Switzerland*

Abstract

The 18th EURO Summer/Winter Institute (ESWI XVIII) took place during the spring 2000 in Switzerland. The topic of ESWI XVIII, "Meta-heuristics in Combinatorial Optimization", was selected due to its great current scientific interest: indeed, in recent years, several meta-heuristics have proved to be highly efficient for the solution of difficult combinatorial optimization problems. The Institute was focused more particularly on the development and the use of local search and population search algorithms. Applications of these meta-heuristics on academic or real life problems were also discussed.

This special issue of EJOR contains papers written by the participants to ESWI XVIII. These papers have benefited from fruitful discussions among the participants, the organizers and the invited speakers. We have tried to summarize here below some guidelines that should help in the design of successful adaptations of meta-heuristics to difficult combinatorial optimization problems.

Keywords: Meta-heuristics; Local search methods; Population search methods

1. Introduction

Combinatorial optimization plays an important role in decision-making since optimal decisions often depend on a non-trivial combination of various factors. Most combinatorial optimization problems are NP-hard, and sharp bounds on the optimal value are typically hard to derive. This means that partial enumeration based exact algorithms have a slow convergence rate, and they can solve only small instances to optimality. But real-life combinatorial optimization problems are typically of big size, and since exact approaches are inadequate, heuristics are commonly used in practice.

There has been a steady evolution over the past forty years in the development of heuristics, which produce solutions of reasonably good quality in a reasonable amount of time. The first proposed heuristics tried to systemize decision-making processes done by hand. With the help of computers which can test a huge amount of combinations in a short amount of time, solutions could easily be generated which turned to be of much better quality when compared to what an expert in the

* Corresponding author.
E-mail addresses: alain.hertz@gerad.ca (A. Hertz), marino.widmer@unifr.ch (M. Widmer).

field could produce by hand. In these early heuristics, much of the emphasis was put on quickly obtaining a feasible solution and possibly applying to it a post-optimization procedure.

Over the last 15 years much of the research effort has concentrated on the development of meta-heuristics, using mainly two principles: *local search* and *population search*. In local search methods, an intensive exploration of the solution space is performed by moving at each step from the current solution to another promising solution in its neighbourhood. Simulated annealing [7], tabu search [3] and variable neighbourhood search [8] are the most famous local search methods. Population search consists of maintaining a pool of good solutions and combining them in order to produce hopefully better solutions. Classical examples are genetic algorithms [5] and adaptive memory procedures [9].

Meta-heuristics are general combinatorial optimization techniques, which are not dedicated to the solution of a particular problem, but are rather designed with the aim of being flexible enough to handle as many different combinatorial problems as possible. These general techniques have rapidly demonstrated their usefulness and efficiency in solving hard problems. Success stories are reported in many papers. While meta-heuristics can handle in theory any combinatorial optimization problem, it is often the case that an important effort must be put on finding the right way to adapt the general ingredients of these methods to the particular considered problem.

We think that in order to be successful in the adaptation of a meta-heuristic to a combinatorial optimization problem, it is necessary to follow some basic principles. We give in the next sections some guidelines which may help in producing such successful adaptations of local search and population search methods for the solution of difficult combinatorial optimization problems.

2. Guidelines for adaptations of local search methods

Let S be a set of solutions to a particular problem, and let f be a cost function that measures the quality of each solution in S. The *neighbourhood* $N(s)$ of a solution s in S is defined as the set of solutions which can be obtained from s by performing simple modifications. Roughly speaking, a local search algorithm starts off with an initial solution in S and then continually tries to find better solutions by searching neighbourhoods. A local search process can be viewed as a walk in a directed graph $G = (S, A)$ where the vertex set S is the set of solutions and there is an arc (s, s') in A if and only if s' is in $N(s)$. By considering the cost function as an altitude, one gets a topology on $G = (S, A)$.

The efficiency of a local search method depends mostly on the modeling. A fine tuning of parameters will never balance a bad definition of the solution set, of the neighbourhood, or of the cost function. We give in this section some general guidelines for successful adaptations of local search methods.

It should be easy to generate solutions in S (a)

It is not rare that finding a feasible solution to a combinatorial optimization problem is an NP-hard problem. In such a case, it would be a very bad idea to define S as the set of feasible solutions to the considered problem since getting an initial solution would be a complex task, and moving from a solution to a neighbour one would not be easier. For such problems, the search space S should be defined by relaxing some constraints of the original problem, and by adding a component in the cost function that penalizes violations of constraints. As an example to this first principle, consider the course-timetabling problem. The main constraint in this kind of problem requires that courses given by the same teacher, or having students in common do not overlap. It is typically very difficult to generate timetables that do not contain any overlapping situation. By relaxing these constraints, it becomes much easier to produce timetables and neighbour solutions can easily be obtained by moving a course to another period, even if such a move induces many new overlapping situations [4].

For each solution s in S, the graph $G(S,A)$ should contain a path linking s to an optimal solution s^* (b)

If the search process visits a solution that does not satisfy this condition, then an optimal solution will never be reached. As an illustration, consider the vehicle routing problem in which a fixed number m of vehicles with limited capacity are required to visit customers in order to fulfill known customer requirements. Assume that a solution is defined as a set of m routes which satisfy all customer requirements as well as the capacity constraints, and suppose that a neighbour solution is obtained by either moving a customer from one route to another, or by permuting two clients from different routes. Such a choice for S violates condition (a) since finding a solution in S is equivalent to solving a bin packing problem. Moreover, this adaptation can also violate condition (b). Indeed, assume there are three vehicles which can each transport 4 units of some product, and assume there are four customers with a demand of 2 units, and four customers with a demand of 1 unit. If the first vehicle visits the four customers with a demand of 1 unit, while the two other vehicles both visit two customers with a demand of 2 units, then no move can be performed to modify the first route without violating the capacity constraints. However, the optimal solution might be one in which the first two vehicles both visit two customers with a demand of 1 unit and one customer with a demand of 2 units, while the third vehicle visits two customers with a demand of 2 units. Conditions (a) and (b) can easily be satisfied by relaxing the capacity constraints [2].

The solutions in $N(s)$ should be in some sense close to s (c)

One could theoretically define $N(s)$ as the set containing all solutions $s' \neq s$ in S. Such a neighbourhood induces a complete graph $G(S,A)$ in which condition (b) is trivially satisfied, and it is possible to move from any initial solution to any optimal one in one step. However, $N(s)$ is so big that finding such an optimal move is a task which is as difficult as the original problem. It is important to define neighbourhoods $N(s)$ in which it is possible to determine the best solution within a reasonably small amount of time. This goal can be achieved by defining $N(s)$ as the set of solutions obtained by performing a simple modification on s. The neighbourhood $N(s)$ of s then contains solutions which are in some sense close to s, and it is often possible to compute the value of a neighbour s' of s by determining the cost difference between s and s', instead of re-computing the value of s' from scratch. For example, consider again the vehicle routing problem, and assume that a neighbour solution s' of s is obtained by moving a customer C from a route R_1 to a route R_2. The cost difference between s and s' can easily be obtained by computing the saving induced by the removal of C from R_1, and by computing the cost induced by the insertion of C into R_2. There is an additional advantage of defining neighbours $s' \in N(s)$ by simple modifications on s: if a neighbour s' has a better cost than s, this may indicate that the arc (s,s') belongs to a short path in $G(S,A)$ from s to an optimal solution, and it becomes then possible to guide the search towards an optimal solution.

The topology induced by the cost function on $G(S,A)$ should not be too flat (d)

As explained above, the cost function can be considered as an altitude, and it therefore induces a topology on $G = (S,A)$ with mountains, valleys and plateaus. It is difficult for a local search to escape from large plateaus since any solution that is not in the boarder of such a plateau has the same cost value as its neighbours, and it is therefore impossible to guide the search towards an optimal solution. A common way to avoid this kind of topology on $G(S,A)$ is to add a component to the cost function which discriminates between solutions having the same value according to the original cost function. As an example, consider the job shop scheduling problem where operations have to be ordered on machines, such that the maximal completion time of all operations, called makespan, is minimized. For a solution s, let $C_k(s)$ denote the time at which all operations are completed on machine k. Then the cost $f(s)$ of solution s (i.e., its makespan) is equal to max $\{C_1(s), C_2(s), \ldots, C_m(s)\}$. If two solutions have the same makespan, then the second largest value in $\{C_1(s), C_2(s), \ldots, C_m(s)\}$ can help discriminating between them. Another possibility is to minimize $C_1(s)^2 + C_2(s)^2 + \cdots + C_m(s)^2$ in order to penalize solutions having too many machines whose completion time is close to the makespan [6].

3. Guidelines for adaptations of population search methods

Population search methods are iterative solution techniques that handle a population of individuals and make them evolve according to some rules that have to be clearly specified. At each iteration, periods of *self-adaptation* alternate with periods of *co-operation*. Self-adaptation means that individuals evolve independently while co-operation implies an exchange of information among the individuals. Many different algorithms can be described within this framework. For example, the selection and crossover operators of genetic algorithms can be seen as co-operation procedures while the mutation operator is part of the self-adaptation process.

As for local search methods, the efficiency of a population search depends mostly on the modeling. We give in this section some general guidelines for successful adaptations of population search methods.

Pertinent information should be transmitted during the co-operation phase (e)

In the co-operation phase, groups of individuals exchange pieces of information and new offspring solutions are created that should combine the best features of the parent solutions. This means that the information that is transmitted during the co-operation phase should be pertinent. Consider for example the k-colouring problem where a colour in $\{1, \ldots, k\}$ must be assigned to each vertex of a graph G so that there are as few edges as possible in G having both endpoints with the same colour, such edges being called *conflicting edges.* An offspring solution s'' can be created from two parent solutions s and s' by colouring some vertices (chosen at random) as in s, and the others as in s'. However, such a combination operator typically produces results of poor quality. Indeed, all colours are equivalent up to a permutation, which means that the colour of a vertex is not a pertinent information. A more useful information is the fact that pairs or subsets of vertices have the same colour. A solution to the k-coloring problem is in fact a partition of the vertex set into k subsets, called *colour classes*, and the aim is to determine a partition so that as few edges have both endpoints in the same colour class. Nowadays, the best population search algorithms for the k-coloring problem create offspring solutions by copying colour classes in parent solutions [1].

The combination of two equivalent parent solutions should not produce an offspring that is different from the parents (f)

Since an offspring solution receives information from parent solutions, it is reasonable to expect that the combination of two parent solutions containing equivalent information produces an offspring which is also equivalent to its parents. To illustrate a situation where condition (f) is not satisfied, consider again the k-coloring problem, and let G be a graph with four vertices v_1, v_2, v_3, v_4 and three edges v_1v_2, v_2v_3, and v_3v_4. Let s be the 2-coloring of G in which v_1 and v_3 have colour 1 and v_2 and v_4 have colour 2, and let s' be the 2-coloring of G obtained from s by permuting colours 1 and 2. Solutions s and s' can be considered as equivalent since they correspond to the same colouring, up to a permutation of the two colours. If one now copies the colours of v_1 and v_2 in s and those of v_3 and v_4 in s', one gets a solution s'' in which vertices v_1 and v_4 have colour 1 and vertices v_2 and v_3 have colour 2. Solution s'' is totally different from the two equivalent parent solutions s and s' since both s and s' are optimal solutions (i.e., they have no conflicting edge), while v_2v_3 is a conflicting edge in s''. This example shows once again that it is a bad idea to create an offspring solution s'' from two parent solutions s and s' by colouring some vertices (chosen at random) as in s, and the others as in s'.

Consider now the problem of finding the smallest integer k for which there exists a k-coloring without conflicting edge. One can define a solution as an ordering of the vertices: each ordering π can be transformed into a colouring by means of a procedure that sequentially colours the vertices according to the ordering defined by π, always giving the smallest possible colour (i.e., the smallest positive integer which is not yet used in the neighbourhood of the considered vertex). For example, both orderings $s = v_1 < v_2 < v_3 < v_4$ and $s' = v_3 < v_4 < v_1 < v_2$ of the above graph G produce a 2-coloring in which v_1 and v_3 have colour 1 and v_2 and v_4 have colour 2. These two orderings can therefore be considered as equivalent. An offspring can then be created by placing some vertices in the same position as in one of the parent solutions, and by placing the other vertices according to the ordering in the second parent solution. For example, assume that an offspring is created from s and s' by copying the positions of v_1 and v_3 in s. The offspring is then $s'' = v_1 < v_4 < v_3 < v_2$, which corresponds to a 3-coloring where vertices v_1 and v_4 having colour 1, vertex v_3 has colour 2, and vertex v_2 having colour 3. Hence, a 3-coloring is obtained by combining two equivalent 2-colorings. Such a situation should definitely be avoided.

Diversity should be preserved in the population (g)

One of the major difficulties observed when using population search algorithms is the *premature convergence* of the process, all solutions in the population having a natural tendency to become equal to the best solution in it. If this occurs, then the population search behaves more or less like a local search since there is nothing to gain in combining equivalent solutions. In order to prevent such a phenomenon, it is important to implement operators that preserve diversity in the population. The mutation operator in genetic algorithms is an example of such a tool, but it usually generates random outputs, which is not necessarily the best thing to do. A technique that helps avoiding premature convergence is to forbid the introduction of a solution in the population if too many 'similar' solutions already exist in it. But this requires the development of measures of similarity that properly detect when there is a danger of premature convergence.

4. Conclusion

It is not possible to provide a general scheme for the adaptation of a local search or a population search to a combinatorial optimization problem. We tried in these few pages to give some guidelines that should help in the design of successful adaptations.

This special issue of EJOR contains papers written by the participants to ESWI XVIII. Each participant had one hour and a half to describe a preliminary paper and his research on the use of meta-heuristics in combinatorial optimisation. The final version of these papers have benefited from fruitful discussions which took place between the participants, the organizers, and the invited speakers (Martin Grötschel and Manuel Laguna), as well as from the above guidelines.

As a conclusion, we want to mention that the fantastic work atmosphere during ESWI XVIII has given the energy to the participants to create a new EURO working group called EU/ME (to be pronounced You and Me): EUropean Chapter on Meta-heuristics.

References

[1] P. Galinier, J.K. Hao, Hybrid evolutionary algorithms for graph coloring, Journal of Combinatorial Optimization 3 (1999) 379–397.

[2] M. Gendreau, A. Hertz, G. Laporte, A tabu search heuristic for the vehicle routing problem, Management Science 40 (1984) 1276–1290.

[3] F. Glover, Future paths for integer programming and links to artificial intelligence, Computers & Operations Research 13 (1986) 433–549.

[4] A. Hertz, Finding a feasible course schedule using tabu search, Discrete Applied Mathematics 35 (1992) 255–270.

[5] J.H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, MI, 1975.

[6] A. Hertz, M. Widmer, An improved tabu search approach for solving the job shop scheduling problem with tooling constraints, Discrete Applied Mathematics 65 (1996) 319–346.

[7] S. Kirkpatrick, C.D. Gellatt Jr., M.P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.

[8] N. Mladenovic, P. Hansen, Variable neighbourhood search, Computers & Operations Research 34 (1997) 1097–1100.

[9] Y. Rochat, E.D. Taillard, Probabilistic diversification and intensification in local search for vehicle routing, Journal of Heuristics 1 (1995) 147–167.