The Design of Memetic Algorithms for Scheduling and Timetabling Problems

E.K. Burke and J.D. Landa Silva

Automated Scheduling, Optimisation and Planning Research Group
School of Computer Science and IT, The University of Nottingham, UK
(ekb|jds)@cs.nott.ac.uk

Summary. There are several characteristics that make scheduling and timetabling problems particularly difficult to solve: they have huge search spaces, they are often highly constrained, they require sophisticated solution representation schemes, and they usually require very time-consuming fitness evaluation routines. There is a considerable number of memetic algorithms that have been proposed in the literature to solve scheduling and timetabling problems. In this chapter, we concentrate on identifying and discussing those strategies that appear to be particularly useful when designing memetic algorithms for this type of problems. For example, the many different ways in which knowledge of the problem domain can be incorporated into memetic algorithms is very helpful to design effective strategies to deal with infeasibility of solutions. Memetic algorithms employ local search, which serves as an effective intensification mechanism that is very useful when using sophisticated representation schemes and time-consuming fitness evaluation functions. These algorithms also incorporate a population, which gives them an effective explorative ability to sample huge search spaces. Another important aspect that has been investigated when designing memetic algorithms for scheduling and timetabling problems, is how to establish the right balance between the work performed by the genetic search and the work performed by the local search. Recently, researchers have put considerable attention in the design of self-adaptive memetic algorithms. That is, to incorporate *memes* that adapt themselves according to the problem domain being solved and also to the particular conditions of the search process. This chapter also discusses some recent ideas proposed by researchers that might be useful when designing self-adaptive memetic algorithms. Finally, we give a summary of the issues discussed throughout the chapter and propose some future research directions in the design of memetic algorithms for scheduling and timetabling problems.

1 Introduction

It is possible to think of a memetic algorithm as an evolutionary algorithm that incorporates knowledge about the problem domain being solved (see [29, 33]). This knowledge can be in the form of specialised operators, heuristics and

other *helpers* that contribute towards a self-improvement ability in the individuals of the population. Most memetic algorithms described in the literature are a combination of genetic algorithms with local search heuristics and these approaches are also known as genetic local search, hybrid genetic algorithms, hybrid evolutionary algorithms and other names (e.g. [13, 22, 25, 38, 44]). This type of hybrid approach has been applied to a vast number of optimisation problems with considerable success (see [34] for a list of example references).

In this paper we concentrate on the application of memetic algorithms to scheduling and timetabling problems such as machine scheduling, educational timetabling and personnel rostering [47]. One goal here is to identify and discuss those strategies that appear to be applied more frequently by researchers and practitioners, when designing memetic algorithms for the type of problems mentioned above. Another goal is to discuss some recent ideas proposed by researchers in this area that might help to design more robust memetic algorithms for scheduling and timetabling problems. We do not attempt to provide an exhaustive survey of memetic approaches to such problems. First, in Sect. 2 we briefly describe the paradigm behind memetic search followed by a short discussion of scheduling and timetabling problems in Sect. 3. Then, in Sect. 4 we concentrate on those strategies that are frequently employed when implementing memetic algorithms for scheduling and timetabling (and perhaps related problems). Scheduling and timetabling problems are usually highly constrained and one of the difficulties when solving these problems is how to deal with infeasibility. Section 4.1 discusses the strategies employed for this purpose within the context of memetic algorithms. For most scheduling and timetabling problems, a complete evaluation of the solution fitness is computationally costly. Therefore, the use of approximate evaluation routines can help to implement more efficient memetic algorithms. This topic is discussed in Sect. 4.2. Selecting an appropriate solution encoding is essential when designing memetic algorithms because both genetic and local search should operate on this encoding in an efficient way. This is discussed in Sect. 4.1. The importance of obtaining knowledge of the fitness landscape to aid the design of better memetic algorithms is discussed in Sect. 4.4. Another very important aspect when designing memetic algorithms is to establish a good balance between the genetic search and the local search parts of the algorithm and this is the subject of Sect. 4.5. Most implementations of memetic algorithms incorporate *memes* designed a priori and these remain unchanged during the search. However, one of the original ideas that motivated the conception of memetic algorithms is the *evolution of memes* and this is discussed in Sect. 4.6. Section 5 describes some local search strategies that have been proposed recently by researchers and that might also be useful if they are incorporated within memetic algorithms. Finally, this paper presents some final remarks and suggests some future research directions in Sect. 6.

2 Memetic Algorithms

It is generally believed that memetic algorithms are successful because they combine the explorative search ability of recombinative evolutionary algorithms and the exploitive search ability of local search methods. An analogy is that the evolutionary part of a memetic algorithm attempts to simulate the genetic evolution of individuals through generations, while the local search part attempts to simulate the individual learning within a lifetime. The majority of memetic algorithms proposed in the literature are a result of incorporating some form of local search to a genetic algorithm. This is illustrated in Fig. 1.

This local search can be for example, constructive heuristics, repair methods, specialised self-improvement operators, etc. The local search phase can be applied before, after or in between the genetic operations. However, as discussed in [29], the interaction between the *memes* and the *genes* can be even more sophisticated than that and most implementations of memetic algorithms fail to reflect the complex interactions of the memetic paradigm. Krasnogor [29] argues that in a truly memetic system:

1. *Memes* also evolve representing the way in which "individuals learn, adopt or imitate certain memes or modify other memes" and,
2. the distribution of *memes* changes dynamically within the population representing the effects of "teaching, preaching, etc." within the population of individuals.

Krasnogor also proposed a formulation of memetic algorithms that attempts to better represent the memetic paradigm including adaptive and self-adaptive *memes*, for more details see [29]. For a more detailed discussion of memetic algorithms see [29, 33, 34] and the references therein.

3 Scheduling and Timetabling Problems

Broadly speaking, the task in scheduling and timetabling problems is to accommodate a set of entities (for example, events, activities, people, vehicles, etc.) into a pattern of time-space so that the available resources are utilised in the best possible way and the existing constraints are satisfied. This paper is mainly concerned with three types of scheduling problems: machine scheduling, educational timetabling and personnel scheduling (also called rostering). However, we will also allude to papers which consider other scheduling problems. A brief description of machine scheduling, educational timetabling and personnel scheduling follows. For more details please see the given references. In machine scheduling, the problem is to schedule a set of jobs for processing on one or more machines [37]. Educational timetabling refers to the allocation

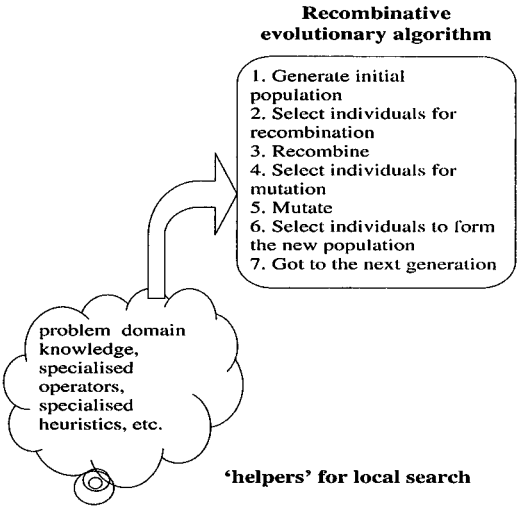

Fig. 1. Most memetic algorithms are a combination of genetic algorithms and local search strategies.

of events (teaching sessions, exams, lab sessions, etc.) to time slots and possibly to rooms [42]. In personnel scheduling, employees must be accommodated into shift patterns [19].

Scheduling and timetabling problems arise in many situations and hence, there is a need for developing effective and efficient automated solution methods. But as with many other combinatorial optimisation problems, scheduling and timetabling are difficult problems to tackle with computer algorithms. The following characteristics are those that make scheduling and timetabling problems very difficult:

Huge search space. The combinatorial nature of scheduling and timetabling problems implies that the size of the search space increases dramatically with the size of the problem making it practically impossible to explore all solutions but for very small problems.

Highly constrained. It is commonly the case that a considerable number of constraints exist in these problems. Constraints limit the possible ways in which a schedule can be constructed. Some constraints must be satisfied for the solutions to be feasible (hard constraints) while other constraints are desirable but not absolutely essential (soft constraints). Examples of constraints are: job A must be processed before job C but after job D (machine scheduling); lecturers cannot teach more than two consecutive sessions (educational timetabling); a minimum number of nurses must be scheduled during busy times (personnel scheduling).

Difficult to represent. It is often difficult to find a representation with associated data structures that capture all details of the problem including the complete set of constraints. Most of the times the problem is simplified, otherwise very elaborate representations are required to model it (see the sections below).

Time-consuming fitness evaluation. Computing the fitness of solutions in scheduling and timetabling problems is usually time consuming. The main reason for this is the existence of many constraints. When a solution is modified even slightly during the search, a number of constraints might be affected and therefore, a complete computation of the whole solution is required.

There is a significant school of thought which says that for most scheduling and timetabling problems, the improvement of solutions can be achieved more effectively with local search heuristics than with recombinative operators. These local search heuristics are usually tailored to the application problem by incorporating knowledge of the problem domain in order to deal with the constraints in a more effective way. It is also generally believed that, because memetic algorithms operate on a population of solutions, they are less dependant on the quality of the initial solutions than local search methods which operate on a single solution. Then, the appeal of applying memetic algorithms for scheduling and timetabling problems is that the powerful improving mechanism of local search is maintained and at the same time, enriched by the addition of a population of individuals.

4 Designing Memetic Algorithms

There is a considerable number of applications of memetic algorithms to scheduling problems reported in the literature including: machine scheduling (e.g. [22, 24]), educational timetabling (e.g. [2, 5, 6, 16, 36]), personnel scheduling (e.g. [1, 9, 26]), maintenance scheduling (e.g. [12, 13]) among many others.

As mentioned above, the addition of local search *helpers* into genetic algorithms is the most common approach reported in the literature. The variety of helpers that have been proposed range from the use of tailored chromosome representations and operators (e.g. [17, 26]) and simple repairing methods based on constraint-based reasoning (e.g. [16]) to very sophisticated combinations of algorithm components in which a memetic algorithm is embedded into a genetic algorithm (e.g. [3]). In this Sect., we discuss the strategies that have been used more frequently by researchers and practitioners when designing memetic algorithms for scheduling and timetabling problems.

4.1 Dealing with Infeasibility

A major issue of concern in scheduling and timetabling is how to deal with the infeasibility of solutions. Due to the large number of hard constraints that typically exist in these problems, generating feasible solutions and keeping them feasible during the search is a difficult task. In general, the first decision that has to be made is whether to consider or not infeasible solutions as part of the searching process. In both cases, the design of adequate solution representations and search operators is an essential ingredient for an effective and efficient operation of the search algorithm. The incorporation of knowledge of the problem domain in the form of choosing the representation and operators according to the existing constraints in the problem, can be considered to be a memetic approach in itself if we accept the broad description of memetic algorithms [33].

Elaborate Encodings and Operators

One way to deal with infeasible solutions is to forbid them completely. That is, only feasible solutions are generated in the initialisation phase and then, the genetic and local search operators are restricted to work only in the feasible region. This approach was used by Burke et al. in the nurse scheduling problem [9] and by Erben and Keppler in the course timetabling problem [18]. Very elaborate solution encodings can also be used to avoid the generation of infeasible solutions altogether. For example, Erben applied grouping genetic algorithms to examination timetabling problems in [17]. In a grouping genetic algorithm, the chromosome representation is made of groups of genes (a group can be, for example, the events scheduled in the same time slot). Then, while in a direct representation of a timetable each gene represents one event, in a grouping representation each gene represents a group of events. In this way, it is easier to design genetic operators that recombine timetables without destroying the feasibility of solutions (i.e. in this case keeping events in the same time slot). Similarly, Kawanaka et al. employed a very elaborate encoding that guaranteed the feasibility of solutions for the nurse scheduling problem [26]. Also, Aickelin and Dowsland [1] implemented two levels of crossover operators for the nurse scheduling problem, one operating at the individual nurse schedule level and another operating at a higher level for the complete schedule. In the first operator, the genes are the time slots while in the second operator the genes are the whole individual schedules of the nurses.

Elaborate encodings and operators help to maintain good sub-solutions (parts of schedules) and aim to generate better complete schedules by mixing good building blocks. However, this can generate more design issues which need to be solved. For example, how do we measure fitness in each type of operator? This might then lead on to investigating various selection schemes within the same algorithm while it is necessary to somehow preserve the good parts of solutions that are already created in order to obtain sensible results,

we should keep in mind that restricting the search to only the feasible regions of the solution space could considerably limit the explorative ability of the memetic algorithm.

Repair Methods and Penalty Schemes

Another approach to deal with infeasibility is to penalise and/or repair infeasible solutions. That is, if the solution encoding and associated search operators permit the generation of infeasible solutions, then repairing heuristics that recover the feasibility of solutions can be implemented. The reparing method should be easy to implement so that no excessive overhead is added to the search process. Also, if the reparing method is too elaborate, it may happen that most of the changes made by the search operators to obtain the new solution from the previous one are reversed by the repairing method resulting in a very inefficient process. For example, repairing heuristics were used by Colorni et al. in the application of genetic and memetic algorithms to high school timetabling problems [14].

An alternative strategy when infeasible solutions are allowed, it is to heavily penalise them in order to discourage their survival. The selection of the penalties must be carefully made. The recommendation is that the penalties should discourage the inclusion of infeasible solutions but without completely eliminating them because infeasible solutions may be required for the algorithm to have a better explorative ability. In our experience, if local search is used, relatively low penalties for infeasibility should be set in many cases because if these penalties are too high, then the local search attempts to recover feasibility first and this could increase substantially the violation of soft constraints [7, 9]. Penalties can be fixed or they can be adapted during the search. Aickelin and Dowsland implemented an adaptive scheme where the infeasibility penalty depends on the number of violated hard constraints [1]: if $q > 0$ then $g_{demand} = \alpha \times q$ where q is the number of violated hard constraints, α is a severity parameter and g_{demand} is the infeasibility penalty weight used in the fitness function.

Multi-Phased Strategies

Some researchers have employed multi-phased approaches to deal with hard constraints and hence, infeasibility of solutions by dividing the solution process in multiple stages. In the first phase the goal is to generate semi-complete feasible solutions. For example, in the nurse scheduling problem, the emphasis can be on the generation of a semi-complete schedule in which it is guaranteed that there are enough nurses available to meet the requirements in each shift, without assigning actual working time slots to each nurse [1]. Then, in the subsequent phases the schedule is incrementally completed by assigning working time slots to individual nurses and forbidding the violation of hard constraints.

Burke and Newall also used a multi-phased approach in their memetic algorithm for the examination timetabling problem [5] (this is an improved version of their previous algorithm presented in [6]). Their memetic approach, however, did not use recombination operators, only mutation operators followed by a hill-climbing algorithm. They constructed a partial timetable and then iteratively applied a memetic algorithm with the aim of scheduling a subset of events in each iteration, i.e. the memetic approach is applied to a subproblem in each iteration. This can be represented in diagrammatic form as shown in Fig. 2.

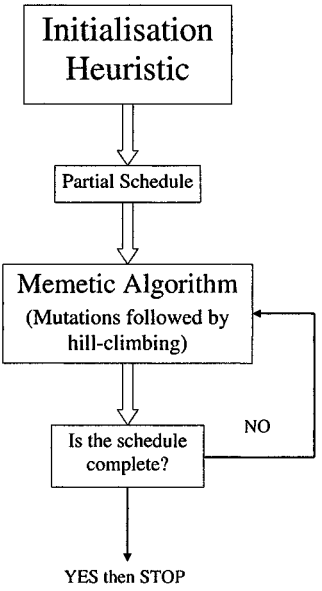

Fig. 2. The multi-phase memetic algorithm implemented by Burke and Newall [5] acts as a peckish (not too greedy) constructive heuristic.

4.2 Approximate Fitness Evaluation

It has been shown that using approximate evaluation (also known as delta evaluation) to measure the quality of solutions helps to improve the efficiency of the search algorithm in problems for which the fitness evaluation function is very time consuming (e.g. [5, 6, 20, 31, 46]). With approximate evaluation, instead of a complete and accurate evaluation of each newly generated solution, only the difference between the previous solution and the new one is computed. Approximate evaluation can be applied as follows to reduce the computation time spent by the memetic algorithm. Two fitness evaluation routines are implemented. One routine completely evaluates the fitness of the new solution (*complete evaluator*). The second one only makes an estimation

of the new fitness based on the previous solution and the changes made (*approximate evaluator*). Suppose that a population of new solutions is created in each generation. Instead of computing the fitness of these solutions with the *complete evaluator*, their fitness is measured with the *approximate evaluator*. Then, only a fraction of the population of new solutions is re-assessed with the *complete evaluator*. These solutions can be for example, the best ones, those that have a minimum quality level, or only those that represent an improvement with respect to the previous solutions. This simple strategy can save a considerable amount of computation time because it is very likely that few solutions would need to be re-evaluated with the *complete evaluator* in the first stages of the evolutionary process. As the search progresses and the overall quality of the population improves, more solutions will have high fitness and perhaps, only the *accurate evaluator* will be used. For more details on how this kind of strategy was implemented for the warehouse scheduling problem see [46], for the examination timetabling problem see [5, 6] and for the space allocation problem see [31].

Of course, for this strategy to be effective it is required that the structure of the combinatorial optimisation problem is such that the quality of the new solutions can be updated by evaluating only the changes made to the previous solution. Approximate evaluation can be applied in any of the stages of a memetic algorithm, i.e. during the local search phase or during the genetic search phase. As mentioned above, a common characteristic of many real world scheduling and timetabling problems is that a considerable number of constraints exist in these situations. Hence, it is very likely that many of the constraints in a particular problem instance are affected by even simple changes to the previous solution. This means that the degree of inaccuracy when using the *approximate evaluator* can be higher than in less constrained combinatorial problems and the implementation of the approximate routine must be carried out carefully.

4.3 Encodings Based on Linked Lists

Another aspect that must be considered when using approximate evaluation is that the solution encoding selected and associated data structures should allow an efficient implementation of the approximate evaluation routine (i.e. much more efficiently that the complete evaluation). In this respect, for scheduling and timetabling problems (and combinatorial problems in general) solution encodings and data structures based on linked lists have been shown to be very helpful for implementing moves and fitness evaluation routines more efficiently (e.g. [5, 31, 39]). This type of representation is advantageous in combinatorial optimisation for several reasons:

- Linked lists can dynamically shrink/grow easily by deleting/adding elements,
- they can also be modified efficiently by only changing pointers,

- linked lists can be used to represent virtually any structure (array, matrix, set, etc.), and
- local search operators such as move, swap, invert, add, delete, change, etc. can be performed directly and very easily with linked lists.

An example of this type of encoding is shown in Fig. 3 for the space allocation problem. In this problem, a set of entities (staff, lecture rooms, labs, etc.) must be allocated to a set of available rooms in such a way that all hard constraints are satisfied, the space misuse is minimised, and the violation of soft constraints is minimised (see [31] for more details). In the encoding of Fig. 3, the lists *Entities*, *Rooms* and *Constraints* hold details of the problem being solved, then these lists remain unchanged throughout the search. That is, these lists hold, for example, the required space for each entity, the capacity of rooms, the type and nominal penalty of constraints, etc. The lists *EntityGene*, *RoomGene* and *ConstraintGene*, hold details of a solution or allocation, i.e. fitness statistics, pointers to the problem data, and pointer that define the structure of the solution. In the solution represented in Fig. 3, entity E1 is allocated to room R5, room R2 is empty, entity E3 is not allocated, constraints C3 and C4 apply to entity E5, etc. With this data structure, it is easy to implement local search moves by only changing the appropriate pointers. Similarly, fitness evaluation routines can be performed efficiently because it is easy to identify which constraints have been affected (by walking along the corresponding linked lists) after a change to the solution structure.

4.4 The Fitness Landscape

Another aspect that makes scheduling and timetabling problems difficult to tackle is that, as in most combinatorial optimisation problems, the shape of the fitness landscape usually depends on each particular problem instance. Moreover, the penalties associated to the various soft constraints in the problem can have an effect on the distribution of local optima. For example, consider two soft constraints SC_1 and SC_2 with associated penalties Γ_1 and Γ_2 respectively. Let SC_1 be 'more important' than SC_2 but 'less difficult' to satisfy than SC_2. That is, more solutions are expected to violate the 'less important' SC_2 than SC_1. If $\Gamma_1 >> \Gamma_2$ it is likely that the search will be biased towards finding many very attractive solutions that satisfy SC_1 (the more important one), but improvements in SC_2 are likely to be overlooked. One the other hand, if $\Gamma_1 << \Gamma_2$ improvements in SC_2 will be preferred over improvements in SC_1 and it is likely that the number of attractive solutions satisfying SC_1 will be less than in the previous case. Further, for the same problem instance, different search operators (local search, crossover, mutation, etc.) explore the solution landscape in very different ways. An ideal situation would be that some knowledge of the fitness landscape for a given problem could be available before the search, but this is rarely the case. Even when preliminary examinations are carried out, it turns out that the solution landscape presents very different characteristics for each problem instance [1].

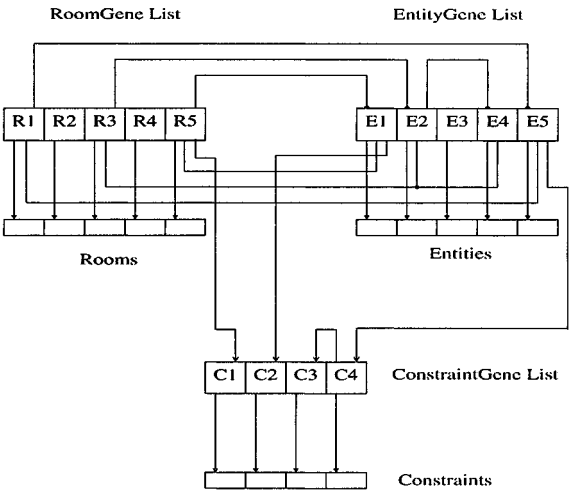

Fig. 3. Solution encoding based on linked lists used for the space allocation problem. The global lists *Entities*, *Rooms* and *Constraints* hold data corresponding to the problem instance being solved. The linked lists of genes *EntityGene*, *RoomGene* and *ConstraintGene*, hold details of a particular allocation or solution.

In this respect, Merz and Freisleben proposed the idea of *fitness landscape analysis* as a means to obtain, a priori, some knowledge of the fitness landscape which could help to design better memetic approaches [32]. They suggest that the first step to perform *fitness landscape analysis* is to define the fitness landscape for the problem instance. For this it is necessary to assign a fitness value to each solution in the search space, i.e. define the fitness function. Then, the spatial structure of the landscape should be defined by defining a metric that measures the distance (in the genotypical space) between two solutions. As Merz and Freisleben note, a simple metric to define the distance between two solutions s and t could be the minimum number of applications of an operator ω required to obtain t from s. Then, they suggest to carry out some preliminary experiments on the problem instance and perform calculations to estimate the properties of the fitness landscape that are known to have an effect on the performance of heuristic methods (see [32] for full details):

- The difference in fitness value between neighbouring solutions.
- The number of local optima.
- An estimation of the distribution of the local optima.
- The topology of the basins of attraction of the local optima.

Knowles and Corne have also carried out some studies on the analysis of the fitness landscape in combinatorial optimisation problems [28] (their approach is discussed below).

4.5 Balance Between Genetics and Memetics

An issue that has received considerable attention when designing memetic algorithms is how to establish the right balance between the work performed by the genetic search and the work performed by the local search. Ideally, in a memetic algorithm, genetic and local search, i.e. the two broadly defined groups of operators, should be able to work together in cooperation instead of against each other [29]. This balance can be tuned from three perspectives among others:

1. The balance between the sophistication of the genetic operators and the local search operators.
2. The decision as to which solutions each group of operators is applied.
3. The balance between the computing time allocated to each type of search.

Some Recommendations

Tuning Genetic and Local Search

For example, with respect the sophistication of the operators, Burke et al. noted that recombining large parts of schedules to form a child solution was very ineffective because the local search heuristics in their memetic algorithm were not powerful enough to improve on these solutions [9]. They observed that instead, it was more beneficial to combine small parts from the parents so that more diversity could be obtained and the local search part could be performed more effectively. But if the local search part of the memetic approach is too powerful, it dominates the search as observed by Burke and Smith when a well tuned tabu search procedure was incorporated as the local search phase in a memetic algorithm for the maintenance scheduling problem [13]. One might feel tempted to use the powerful local search operators and heuristics already known for scheduling and timetabling problems, but the difficulties mentioned above can be encountered.

In the case of which solutions should be applied to each group of operators, an approach that has been used is to improve by local search only a number of the best solutions in the population (e.g. [1]). In [22] Ishibuchi et al. implemented a first version of a memetic algorithm for the flow-shop scheduling problem (named genetic local search in that paper) where they proposed not to examine the whole neighbourhood but only a fraction of it (i.e. best of k instead of best of all) and stop the search when no better neighbour is found after a small number of iterations. Later, they also proposed to apply local search to only good offspring to improve the search ability of

their genetic local search approach [23]. Although in that paper Ishibuchi et al. consider multi-objective flow-shop scheduling problems, this idea can be transferred from the multi-objective domain to the single-objective domain by noting that good solutions are not those that offer a good coverage of the front but solutions that represent a good subset of the population because of fitness and diversity. In a more recent study, Ishibuchi et al. proposed the following strategies [24]:

- To apply local search to a subset of the population selected based upon a given probability p and on the fitness of the solutions according to preset criteria.
- To apply the local search procedure not after each generation but every $T > 1$ generations.
- To carry out preliminary experiments for tuning the values of the above mentioned parameters (k, p and T).
- To carry out preliminary experiments to establish the adequate values for the crossover and mutation probabilities for tuning the genetic search (provided the parameters of k, p and T have been fixed).

With respect to the genetic operators, it has been proposed to apply them only to parent solutions that have a certain distance between them in the genotypical space (this is called a mating restriction) [35]. It has been observed that applying the local search for a limited number of iterations enables better results in the long run as reported by Burke and Smith for the maintenance scheduling problem [12].

Archives of Solutions

The use of archives of solutions (a form of elitism) as in [27] can also be employed to enhance the balance of genetic and local search. Such an archive can be used to store elite solutions from which to chose the appropriate ones before carrying out the genetic operations.

Reacting to the Shape of the Fitness Landscape

Measuring the characteristics of the fitness landscape as the search progresses can help in the design of a dynamic method to balance the genetic and the local search. This has been investigated by Knowles and Corne in the context of the multi-objective quadratic assignment problem [28]. Fitness landscape analysis techniques can help to identify the structure of a given problem, not only before the search but perhaps also dynamically during the search [32]. The study of the fitness landscape is a very promising avenue of research that can be of considerable benefit to enhance the performance of memetic algorithms, particularly on problems such as scheduling and timetabling. This is because adequate operators could then be designed and the search tuned according to the landscape.

Common Annealing Schedules

Krasnogor and Smith have investigated a form of adapting the algorithm performance according to the search, so that the emphasis can be adapted: 1) to improve the fitness of the population, or 2) to diversify the population [30]. They used a common temperature for the whole population to control the acceptance of solutions during the local search phase. The temperature is inversely proportional to the spread of fitness in the population. Therefore, as the population converges (spread of fitness is reduced) the temperature increases and more non-improving solutions are accepted in order to induce more exploration. Once the spread of fitness is recovered, the temperature falls so that only improving solutions are accepted and the search acts as a local search procedure. Burke et al. implemented a population-based annealing algorithm in which a common annealing schedule is used to control the acceptance probability of solutions generated by each of the individuals in the population [7]. In their approach, there is no recombination of solutions and the balance between exploration and intensification is managed only by the evolution of the population by self-improvement. They applied the algorithm to the space allocation problem which shares various features with the class of timetabling problems and was briefly described above (see [31]). The idea behind their common annealing schedule was to allow a certain degree of flexibility in an attempt to use the mechanism as a diversification strategy.

4.6 Towards Adaptive Memes

Most of the research related to memetic algorithms for scheduling and timetabling problems has concentrated on: 1) the design of specialised operators that enable constraints to be dealt with more efficiently, and/or 2) the comparison between the performance of different operators. For example, Alkan and Ozcan recently carried out a comparison of various mutation operators that are directed towards the satisfaction of specific constraints (these can be considered as local search heuristic more than mutation operators, see our discussion in the final Sect.) [2]. Another aspect that Alkan and Ozcan investigated was the comparison of various hill-climbers. They designed a specific hill-climber for each type of constraint and all hill-climbers were combined under a single hill-climber controlling the whole local search phase.

This idea of designing specialised local search heuristics to target a particular constraint or group of constraints has also been investigated by Viana et al. in what they call *constraint oriented neighbourhoods* [45]. Their idea is to use, for a given problem, neighbourhood structures that explicitly take into account the particular characteristics of the problem constraints. Then, during the local search the neighbourhood moves are chosen according to the constraints that are not satisfied in that moment. The adaptation of neighbourhood search heuristics has also been explored by Burke et al. in the

context of multi-objective optimisation where different heuristics are targeted to different objectives (for more details see [11]).

The progress or success rate of different operators can be assessed during the search. Then, their application can be adapted accordingly to the conditions of the search process. For example, Basseur et al. used a scheme to measure the progress of various mutation operators when tackling multi-objective flow-shop scheduling problems [3]. They implemented various mutation operators which are applied with the same probability at the beginning of the search. As the search progresses, the decision as to which mutation operator to use is made dynamically. The generated solutions are evaluated before and after the application of each mutation operator. Depending on the success of the operator, they calculate an average growth value which is used to dynamically adjust the probability of each mutation operator. More specifically:

1. When a mutation operator M is applied, a solution $M(x)$ is generated from a solution x.
2. The progress of the mutation operator M when applied to solution x is 1 if x is dominated by $M(x)$, 0 if x dominates $M(x)$ and 0.5 otherwise. A solution x dominates a solution y if x is as good as y in all objectives and better in at least one of them (see [43]).
3. The average $Progress(M(i))$ of each mutation operator M is calculated by summing all the progresses of M and dividing it by the number of solutions to which M was applied.
4. Then, the probability of each mutation operator is adjusted using (1) where η is the number of mutation operators and δ indicates the minimal ratio value permitted for each operator. That is, δ is a parameter that permits to keep each operator even if the progress of the operator is too poor.

$$P_{M(i)} = \frac{Progress(M(i))}{\sum_{j=1}^{\eta} Progress(M(j))} \times (1 - \eta \times \delta) + \delta \ . \tag{1}$$

As discussed here, although some research has been carried out on how to adapt the application of different genetic and local search operators and heuristics (*memes*) throughout the search, in most cases the *memes* have been designed before the search and remain unchanged during the process. The notion of evolution of *memes* instead of evolution of *genes* in the context of timetabling was first suggested by Ross et al. who said: "we suggest that a GA might be better employed in searching for a good algorithm rather than searching for a specific solution to a specific problem" [40]. As also argued by Krasnogor, the evolution of *memes* is an aspect that deserves more attention in order to design more advanced and improved memetic systems [29].

5 Other Ideas for Memetics

5.1 Cooperative Local Search

As discussed in Sect. 2, most memetic algorithms result from the hybridisation of evolutionary algorithms and the incorporation of a variety of specialised *helpers* such as elaborate encodings, local search heuristics, etc., from the knowledge of the problem domain. Another form of hybridising evolutionary methods and local search is by adding some elements of evolutionary algorithms (such as genetic operators, populations of solutions, a common annealing schedule, etc.) into a *cooperative local search* scheme. This form of hybridisation is illustrated in Fig. 4. To describe the difference between most memetic approaches and *cooperative local search*, we also refer to Fig. 3.

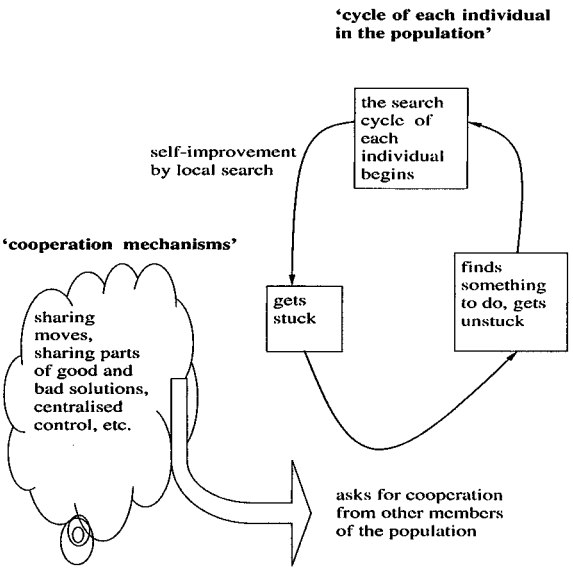

Fig. 4. In a *cooperative local search* scheme, each individual carries out its own local search. When an individual gets stuck it asks for the cooperation of the population in order to find something to do to get unstuck and continue the search from another position in the solution space. The results achieved by each individual may be different at different times and this encourages diversity within the population.

While in most memetic algorithms as depicted in Fig. 3, the structure of the evolutionary algorithm based on generations is maintained, in the *cooperative local search* approach, the self-improving individual cycle is the *driving mechanism* and the *helpers* come from evolutionary methods. This form of hybridisation was proposed in [31] as an alternative strategy to combine the explorative capability of genetic search with the intensification ability of local

searchers. This type of hybrid, can be beneficial in those problems in which the recombination of solutions requires the design of specialised encodings, repairing methods or recombinative operators. This is the case in most of the scheduling and timetabling problems as discussed above. With the *cooperative local search* approach, a population of local searchers evolve mainly by self-improvement. But the individuals also share information during the search process with the rest of the population and hence, a form of recombination can be achieved. In [31] this concept was applied to the space allocation problem and it proved to be very effective. The cooperation between individuals was accomplished by maintaining a pool of good and bad parts of solutions. Then, the cooperation (possible recombination) between individuals in the population is asynchronous as opposed to most memetic algorithms. In a synchronous mechanism (as in memetic algorithms) the cooperation is regulated by generations. In an asynchronous mechanism (as proposed in *cooperative local search*) the individuals cooperate between them at any required time. When an individual gets stuck, it asks for the help of the population and this is implemented by accessing the pool of genes. There are several variants of this approach that can be explored following the terminology of hybrid metaheuristics proposed by Talbi in [44](see also [38]). For example, the cooperation can be synchronous or asynchronous, the explorers can employ the same (homogeneous) or different (heterogeneous) local search procedures and also can search the same or different areas of the solution space (global, partial or functional).

5.2 Teams of Heuristics

Some researchers have investigated the design of search algorithms based upon a collection of simple local search heuristics. The idea is that a set of simple local search heuristics can be applied in a systematic or adaptive way to tackle difficult combinatorial optimization problems. Examples of these approaches are:

Variable neighbourhood search [21]. In variable neighbourhood search a number of different neighbourhood structures are used in a systematic fashion to attempt improvements in the current solution while attempting to avoid getting stuck in poor local optima.

A-teams of heuristics [41]. An A-team of heuristics consists of a set of constructors (to generate solutions), a set of improvers (to perform local search and improve solutions) and a set of destructors (to eliminate poor quality solutions). All these heuristics operate on a population of solutions and all of them behave like independent agents cooperating in an asynchronous fashion.

Hyper-heuristics [10]. A hyper-heuristic can be described as a heuristic that manages the application of a set of heuristics (which can be simple neighbour-

hood heuristics or elaborate meta-heuristics) during the search. For example, a hyper-heuristic might, at each moment during the search, make the decision as to which heuristic to use according to the historical performance of each heuristic. The central idea is that the hyper-heuristic learns and adapts itself dynamically during the search process.

The ideas behind the above approaches can be used to inspire more strategies for designing more advanced memetic algorithms for scheduling, timetabling problems and other combinatorial optimisation problems. Specifically, the self-adaptation of local search heuristics would help to further develop the idea of meme evolution [29].

6 Final Remarks and Future Research

6.1 Scheduling and Timetabling: Interesting Domain for Memetic Algorithms

Scheduling and timetabling problems represent an interesting domain for the application of memetic algorithms. There are already a number of applications reported in the literature (e.g. [1, 2, 3, 5, 6, 9, 12, 13, 16, 19, 22, 26, 36]) but certainly there are still several promising research directions to be explored. We can summarise some of the reasons for which memetic algorithms are a good approach to solve scheduling and timetabling problems as follows:

- The size of the search space in scheduling and timetabling problems is huge for most real-world problems, and a good explorative ability, which memetic approaches have, is required.
- These problems are highly constrained and therefore, most of the times it is easier to self-improve solutions that to recombine them. This highly constrained nature also leads to the design of specialised solution encodings. Memetic algorithms also incorporate specialised encodings and operators for self-improvement of solutions, which are based on the knowledge of the problem domain.
- A complete fitness evaluation function is time consuming in many real-world scheduling and timetabling problems. Using approximate evaluation in memetic algorithms is more robust than in single-solution methods. This is because having a population of new solutions instead of only one new solution, helps to reduce the the effect of the error in the fitness estimation.

6.2 Ideas That Have Been Investigated

In the literature and from the previous sections in this paper, we can summarise some of the ideas that have been investigated with respect to the application of memetic algorithms for scheduling and timetabling problems:

- The design of specialised encodings (e.g. linked lists), local search strategies, genetic operators, and multi-phased methods, all assist the algorithm in dealing with infeasible solutions and have received considerable attention.
- Approximate fitness evaluation has been used in memetic algorithms but to a lesser extent than the above.
- Some work has been carried out on analysing the fitness landscape to inform the design of the memetic algorithm. That is, to select more appropriate operators and to tune the genetic and local search phases. However, this analysis is usually performed prior to the implementation of the memetic approach.
- The balance between the genetic and the local search parts of memetic algorithms has received considerable attention particularly in recent years. The aspects studied here include: the sophistication of operators, the selection of solutions to which apply the operators (including elitist strategies such as archives of solutions), the computing time allocated to the genetic and the local search, the tuning and balance of the parameters (previous to the search), use of population control mechanisms (such as common annealing schedule), etc.

6.3 A Few Thoughts

Designing a memetic algorithm is frequently associated with the incorporation of knowledge from the problem domain in the form of *helpers* to evolutionary algorithms. We should be careful because almost every new piece of specific knowledge that is added to a memetic algorithm can potentially produce improved results. Then, we can many times keep designing a 'new' or an 'improved' version of the memetic algorithm, i.e. an incremental design of algorithms. We should concentrate on the main ideas and strategies without getting lost in the details of the different implementations.

Krasnogor proposes in [29] a grammar to formulate a wide range of memetic algorithms. He also expresses that the grammar can help to envisage many more different implementations of memetic algorithms that have not been investigated. It would be interesting to investigate such variants. But we should also be careful and focus on the main strategies for designing memetic algorithms and not necessarily on the many ways in which they are combined.

6.4 Suggested Future Research

We argue here that by studying in detail the problem domain, there is always room to create a 'new' memetic approach. If memetic algorithms simulate the evolution of ideas, should we not take for granted that many different ideas would exist so we should concentrate more on the mechanism to evolve these ideas instead of manufacturing these ideas by hand before the search process? The research carried out by researchers on the design of memetic algorithms

for scheduling and timetabling has contributed enormously to our understanding of specialised encodings, operators, heuristics, evaluation routines, etc. and their inter-relationships. But as suggested in [29, 33], to learn more about the memetic paradigm, we should concentrate now on using this knowledge for designing approaches to evolve *genes* and *memes* during the search and also automatically select *memes* given the problem domain and also the particular instance characteristics. This was also proposed by Ishibuchi et al. in [24] where they suggested to investigate the dynamic adaptation of the balance between local and genetic search. We also believe that an important challenge in this area is to investigate the design of self-adaptive memetic systems, i.e. the *evolution of genes and memes.*

References

1. Aickelin U., Dowsland K.A. (2000). Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. Journal of scheduling, 3(3), 139–153.
2. Alkan A., Ozcan E. (2003). Memetic algorithms for timetabling. Proceedings of the 2003 congress on evolutionary computation (CEC 2003), 1796–1802, IEEE press.
3. Basseur M., Seynhaeve F., Talbi E.G. (2002). Design of multi-objective evolutionary algorithms to the flow-shop scheduling problem. Proceedings of the 2002 congress on evolutionary computation (CEC 2002), IEEE press, 1151–1156.
4. Blazewicz J., Domschke W., Pesch E. (1996). The job shop scheduling problem: conventional and new solution techniques. European journal of operational research, 93, 1–33.
5. Burke E.K., Newall J.P. (1999). A multi-stage evolutionary algorithm for the timetable problem. IEEE transactions on evolutionary computation, 3(1), 1085–1092.
6. Burke E.K., Newall J.P., Weare R.F. (1996). A memetic algorithm for university exam timetabling. The practice and theory of automated timetabling: Selected papers from the 1st international conference on the practice and theory of automated timetabling (PATAT 1995), Lecture notes in computer science, 1153, 241–250, Springer.
7. Burke E.K., Cowling P., Landa Silva J.D. (2001). Hybrid population-based metaheuristic approaches for the space allocation problem. Proceedings of the 2001 congress on evolutionary computation (CEC 2001), IEEE press, 232–239.
8. Burke E.K., Cowling P., Landa Silva J.D., Petrovic S. (2001). Combining hybrid metaheuristics and populations for the multiobjective optimisation of space allocation problems. Proceedings of the 2001 genetic and evolutionary computation conference (GECCO 2001), Morgan kaufmann, 1252–1259.
9. Burke E., Cowling P., De Causmaecker P., Vanden Berghe G. (2001). A memetic approach to the nurse rostering problem. Applied intelligence, 15(3), 199–214.
10. Burke E.K., Hart E., Kendall G., Newall J., Ross P., Schulemburg S. (2003). Hyper-heuristics: an emerging direction in modern search technology. In: Glover F.W., Kochenberger G.A. (eds.), Handbook of metaheuristics, Kluwer academic publishers, 2003.

11. Burke E.K., Landa Silva J.D., Soubeiga E. (2003). Hyperheuristic approaches for multiobjective optimisation. Proceedings of the 5th metaheuristics international conference (MIC 2003), Kyoto Japan. Extended version available from the authors.
12. Burke E.K., Smith A. (1999). A memetic algorithm to schedule planned maintenance for the national grid. ACM Journal of experimental algorithmics, 4(1), 1084–1096.
13. Burke E.K., Smith A. (2000) Hybrid evolutionary techniques for the maintenance scheduling problem, IEEE transactions on power systems, 15(1), 122–128.
14. Colorni A., Dorigo M., Maniezzo V. (1998). Metaheuristics for high school timetabling, Computational optimization and applications, 9, 275–298.
15. Corne D., Dorigo M., Glover F. (eds.) (1999). New ideas in optimisation. McGraw Hill.
16. Deris S., Omatu S., Ohta H., Saad P. (1999). Incorporating constraint propagation in genetic algorithm for university timetable planning. Engineering applications of artificial intelligence, 12, 241–253.
17. Erben Wilhelm (2001). A grouping genetic algorithm for graph colouring and exam timetabling. The practice and theory of automated timetabling III: Selected papers from the 3rd international conference on the practice and theory of automated timetabling (PATAT 2000), Lecture notes in computer science, 2079, 132-156, Springer.
18. Erben W., Keppler J. (1996). A genetic algorithm solving a weekly course-timetabling problem. The practice and theory of automated timetabling: Selected papers from the 1st international conference on the practice and theory of automated timetabling (PATAT 1995), Lecture notes in computer science, 1153, 198–211, Springer.
19. Ernst A.T., Jiang H., Krishnamoorthy M., Sier D. (2004). Staff scheduling and rostering: a review of applications, methods and models. European journal of operational research, 153, 3–27.
20. Grefenstette J.J., Fitzpatrick M.J. (1985). Genetic search with approximate function evaluation. Genetic algorithms and their applications: Proceedings of the first international conference on genetic algorithms, 112–120.
21. Mlandenovic N., Hansen P. (1997). Variable neighbourhood search. Computers and operations research, 24(11), 1097–1100.
22. Ishibuchi H., Murata T., Tomioka S. (1997). Effectiveness of genetic local search algorithms, Proceedings of the seventh international conference on genetic algorithms, 505–512.
23. Ishibuchi H., Yoshida T., Murata T. (2002b). Selection of initial solutions for local search in multiobjective genetic local search. Proceedings of the 2002 congress on evolutionary computation (CEC 2002), 950–955, IEEE press.
24. Ishibuchi H., Yoshida T., Murata T. (2003). Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. IEEE transactions on evolutionary computation, 7(2), 204–223.
25. Jaszkiewicz A. (2002). Genetic local search for multi-objective combinatorial optimization. European journal of operational research, 137(1), 50–71.
26. Kawanaka H., Yamamoto K., Toshikawa T., Shinogi T., Tsuruoka S. (2001). Genetic algorithm with the constraints for nurse scheduling problem. Proceedings of the 2001 congress on evolutionary computation (CEC 2001), 1123–1130, IEEE press.

27. Knowles J.D., Corne D.W. (2000). M-PAES a memetic algorithm for multiobjective optimization. Proceedings of the 2000 congress on evolutionary computation (CEC 2000), 325–332, IEEE press.
28. Knowles J.D., Corne D.W. (2002). Towards landscape analyses to inform the design of a hybrid local search for the multiobjective quadratic assignment problem. In: Soft computing systems: design, management and applications, 271–279, IOS Press.
29. Krasnogor N. (2002). Studies on the theory and design space of memetic algorithms. PhD thesis, Faculty of computing, engineering and mathematical sciences, University of the West of England, UK.
30. Krasnogor N., Smith J. (2000). A memetic algorithm with self-adaptive local search: TSP as a case study. Proceedings of the 2000 genetic and evolutionary computation conference (GECCO 2000), 987–994, Morgan kaufmann.
31. Landa-Silva J.D. (2003). Metaheuristic and multiobjective approaches for space allocation. PhD Thesis, School of Computer Science and Information Technology, University of Nottingham.
32. Merz P, Freisleben B. (1999). Fitness landscape and memetic algorithm design. In: Corne D., Dorigo M., Glover F. (eds.), New ideas in optimisation, McGraw Hill, 245–260.
33. Moscato P. (1999). Memetic algorithms: a short introduction. In: Corne D., Dorigo M., Glover F. (eds.), New Ideas in Optimisation, 219–234, McGraw Hill, 1999.
34. Moscato P. (2002). Memetic algorithms' home page. Online, available at http://www.densis.fee.unicamp.br/moscato/memetichome.html.
35. Murata T., Ishibuchi H., Gen M. (2000). Cellular genetic local search for multiobjective optimization. Proceedings of the 2000 genetic and evolutionary computation conference (GECCO 2000), Morgan kaufmann, 307–314.
36. Paechter B., Cumming A., Norman M.G., Luchiam H. (1996). Extensions to a memetic timetabling system. The practice and theory of automated timetabling: Selected papers from the 1st international conference on the practice and theory of automated timetabling (PATAT 1995), Lecture notes in computer science, 1153, 251–265, Springer.
37. Pinedo Michael (1995). Scheduling, theory, algorithms, and systems. Prentice-hall.
38. Preux Ph., Talbi E.G. (1999). Towards hybrid evolutionary algorithms. International transactions in operational research, 6, 557–570.
39. Randall M., Abramson D. (2001). A general meta-heuristic based solver for combinatorial optimisation problems. Computational optimization and applications, 20, 185–210.
40. Ross P., Hart E., Corne D. (1998). Some Observations about GA-based exam timetabling. The practice and theory of automated timetabling II: Selected papers from the 2nd international conference on the practice and theory of automated timetabling (PATAT 1997), Lecture notes in computer science, 1408, 115–129, Springer.
41. Salman F.S., Kalagnaman J.R., Murthy S., Davenport A. (2002). Cooperative strategies for solving bicriteria sparse multiple knapsack problem. Journal of heuristics, 8, 215–239.
42. Schaerf A. (1999). A Survey of automated timetabling. Artificial intelligence review, 13, 87–127.

43. Steuer Ralph E. (1986). Multiple criteria optimization: theory, computation and application. Wiley.
44. Talbi E.G. (2002). A Taxonomy of hybrid metaheuristics. Journal of heuristics, 8, 541–564.
45. Viana A., Pinho de Sousa J., Matos M.A. (2003). GRASP with constraint neighbourhoods: an application to the unit commitment problem. Proceedings of the 5th metaheuristics international conference (MIC 2003), 2003.
46. Watson J.P., Rana S., Whitley L.D., Howe A.E. (1999). The impact of approximate evaluation on the performance of search algorithms for warehouse scheduling. Journal of scheduling, 2, 79–98.
47. Wren A. (1996). Scheduling, timetabling and rostering, a special relationship?. The practice and theory of automated timetabling: Selected papers from the 1st international conference on the practice and theory of automated timetabling (PATAT 1995), Lecture notes in computer science, 1153, 46–75, Springer.