Meta-heuristics and Artificial Intelligence

Jin-Kao Hao, Christine Solnon

Meta-heuristics and Artificial Intelligence

Jin-Kao Hao and Christine Solnon

Abstract Meta-heuristics are generic search methods that are used to solve challenging combinatorial problems. We describe these methods and highlight their common features and differences by grouping them in two main kinds of approaches: *Perturbative meta-heuristics* that build new combinations by modifying existing combinations (such as, for example, genetic algorithms and local search), and *Constructive meta-heuristics* that generate new combinations in an incremental way by using a stochastic model (such as, for example, estimation of distribution algorithms and ant colony optimization). These approaches may be hybridised, and we describe some classical hybrid schemes. We also introduce the notions of diversification (exploration) and intensification (exploitation), which are shared by all these meta-heuristics: diversification aims at ensuring a good sampling of the search space and, therefore, at reducing the risk of ignoring a promising sub-area which actually contains high-quality solutions, whereas intensification aims at locating the best combinations within a limited region of the search space. Finally, we describe two applications of meta-heuristics to typical artificial intelligence problems: satisfiability of Boolean formulas, and constraint satisfaction problems.

Jin-Kao Hao
LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France
e-mail: jin-kao.hao@univ-angers.fr

Christine Solnon
LIRIS-CNRS, INSA de Lyon, 20 Avenue Albert Einstein, 69621 Villeurbanne, France
e-mail: christine.solnon@liris.cnrs.fr

1 Introduction

Meta-heuristics are generic methods that may be used to solve complex and challenging combinatorial search problems. These problems are challenging for computer scientists because solving them involves examining a huge number – usually exponential – of combinations. Every man jack has already encountered such a combinatorial explosion phenomenon, which transforms an apparently very simple problem into a tricky brain-teaser as soon as one increases the size of the problem to solve. This is the case, for example, when we try to solve tiling puzzles such as pentaminoes: when the number of tiles is small enough, these problems are rather easily solved by a systematic review of all possible combinations; however, when slightly increasing the number of tiles, the number of different combinations to review increases so drastically that the problem cannot be solved any longer by a simple enumeration and, for larger problems, even the most powerful computer cannot enumerate all combinations within a reasonable amount of time.

The challenge for solving these problems clearly goes beyond puzzles. Indeed, this combinatorial explosion phenomenon also occurs in many industrial problems such as, for example, scheduling activities, planning a production, or packing objects of different volumes into a finite number of bins. Hence, it is most important to design intelligent algorithms that are actually able to solve these hard combinatorial problems in a reasonable amount of time.

There exist three main approaches for tackling combinatorial problems. *Exact approaches* explore the space of combinations (i.e., candidate solutions) in a systematic way until either a solution is found or an inconsistency is proven. In order to (try to) restrain combinatorial explosion, these approaches structure the set of all combinations in a tree and use pruning techniques —to reduce the search space— and ordering heuristics —to define the order in which it is explored. These approaches are able to solve many problems. However, pruning techniques and ordering heuristics are not always able to restrain combinatorial explosion, and some problem instances cannot be solved by these approaches within a reasonable amount of time.

Heuristic approaches get round combinatorial explosion by willfully ignoring some combinations. As a consequence, these approaches may miss the optimal solution and, of course, they cannot prove the optimality of the combination they found even if it is actually optimal. As a counterpart, their time complexity usually is polynomial.

Approximation approaches aim to find approximate solutions with provable guarantees on the distance of the achieved solution to the optimum. If an algorithm can find a solution within a factor α of the optimum for every instance of the given problem, it is an α-approximation algorithm.

Organisation of the chapter

There mainly exist two kinds of heuristic approaches: *Perturbative heuristic approaches* —described in Section 2— build new combinations by modifying existing combinations; *Constructive heuristic approaches* —described in Section 3— generate new combinations in an incremental way by using a (stochastic) model. These approaches may be hybridised, and we describe in Section 4 some classical hybrid schemes. Then, we introduce in Section 5 the notions of diversification (exploration) and intensification (exploitation) which are shared by all these heuristic approaches: diversification aims at ensuring a good sampling of the search space and, therefore, at reducing the risk of ignoring a sub-area which actually contains a solution, whereas intensification aims at guiding the search towards the best combinations within a limited region of the search space. Finally, we describe in Section 6 two applications of meta-heuristics to typical artificial intelligence problems: satisfiability of Boolean formulas (SAT), and constraint satisfaction problems (CSPs).

Notations

In this chapter, we assume that the search problem to be solved is defined by a couple (S, f) such that S is a set of candidate combinations, and $f : S \to \mathbb{R}$ is an objective function that associates a numerical value with every combination of S. Solving such a problem involves finding the combination $s^* \in S$ that optimises (maximises or minimises) f.

We more particularly illustrate the different meta-heuristics introduced in this chapter on the traveling salesman problem (TSP): given a set V of cities and a function $d : V \times V \to \mathbb{R}$ such that for each pair of cities $\{i, j\} \subseteq V$, $d(i, j)$ is the distance between i and j, the goal is to find the shortest route that passes through each city of V exactly once. For this problem, the set S of candidate combinations is defined by the set of all circular permutations of V and the objective function f to be minimised is defined by the sum of the distances between every couple of consecutive cities in the permutation.

2 Perturbative Meta-heuristics

Perturbative approaches explore the combination space S by iteratively perturbating combinations: starting from one or more initial combinations (that can be obtained by any means, often randomly or greedily), the idea is to iteratively generate new combinations by modifying some previously generated combinations. These approaches are said to be *instance-based* in [Zlochin et al, 2004]. The most well known perturbative approaches are genetic algorithms, described in Section 2.1, and local search, described in Section 2.2.

2.1 Genetic Algorithms

Genetic algorithms [Holland, 1975; Goldberg, 1989; Eiben and Smith, 2003] draw their inspiration from the evolutionary process of biological organisms in nature and, more particularly, from three main mechanisms which allow them to better fit their environment:

- *Natural selection* implies that individuals that are well fitted to their environment usually have a better chance of surviving and, therefore, reproducing.
- *Reproduction by cross-over* implies that an individual inherits its features from its two parents in such a way that two well-fitted individuals tend to generate new individuals that also are well-fitted, and hopefully, better fitted.
- *Mutation* implies that some features may randomly appear or disappear, thus allowing nature to introduce new abilities that are spread to the next generations thanks to natural selection and cross-over if these new abilities better fit the individual to its environment.

Genetic algorithms combine these three mechanisms to define a meta-heuristic for solving combinatorial optimisation problems. The idea is to evolve a population of combinations—by applying selection, cross-over and mutation— in order to find better fitted combinations, where the fitness of a combination is assessed with respect to the objective function to optimise. Algorithm 1 describes this basic principle, the main steps of which are briefly described in the next paragraphs.

Algorithm 1: Genetic Algorithm

```
Initialise the population
while stopping criteria not reached do
    Select combinations from the population
    Create new combinations by recombination and mutation
    Update the population
return the best combination that ever belonged to the population
```

Initialisation of the population: In most cases, the initial population is randomly generated with respect to a uniform distribution in order to ensure a good diversity of the combinations.

Selection: This step involves choosing the combinations of the population that will be used to generate new combinations (by recombination and mutation). Selection procedures are usually stochastic and designed in such a way that the selection of the best combinations is favoured while leaving a small chance to worse combinations to be selected. There exist many different ways to implement this selection step. For example, *tournament selection* consists in randomly selecting a few combinations in the population, and keeping the best one (or randomly selecting one with respect to a probability proportional to the objective function). Selection may also consider other criteria, such as diversity.

Recombination (cross-over): This step aims at generating new combinations from selected combinations. The goal is to lead the search towards a new zone of the space where better combinations may be found. To this aim, the recombination should be well-fitted to the function to be optimised, and able to transmit good properties of the parents to the new combination. Moreover, recombination should allow to create diversified children. From a diversification/intensification point of view, recombination has a strategic diversification role, with a long term goal of intensification.

Mutation: Mutation aims at slightly modifying combinations obtained after cross-over. It is usually implemented by randomly selecting combination components and randomly choosing new values to these components.

Exemple 1 *For the TSP, a simple recombination consists in copying a sub-sequence of the first parent, and completing the permutation by sequencing the cities that are missing in the order they occur in the second parent. A classical mutation operator consists in randomly choosing some cities and exchanging their positions.*

Population updating step: This step aims at replacing some combinations of the population by some of the new combinations —that have been generated by applying recombination and mutation operators— in order to create the next generation population. The update policy is essential to maintain an appropriate level of diversity in the population, to prevent the search process from premature convergence, and to allow the algorithm to discover new promising areas of the search space. Hence, decisions are often taken with respect to criteria related to quality and diversity. For example, a well known quality-based update rule consists in replacing the worse combination of the population, while a diversity-based rule consists in replacing old combinations by similar new combinations, with respect to some given similarity measure [Lü and Hao, 2010; Porumbel et al, 2010]. Other criteria such as the age may also be considered.

Stopping criteria: The evolution process is iterated, from generation to generation, until either it has found a solution whose quality reaches some given bound or a fixed number of generations or a CPU-time limit have been reached. One may also use diversity indicators such as, for example, the resampling rate or the average pairwise distance, to restart a new search when the population becomes too uniform.

2.2 Local Search

Local Search (LS) explores the search space by iteratively modifying a combination: starting from an initial combination, it iteratively moves from the current combination to a neighbour combination obtained by applying some transformation to it [Hoos and Stützle, 2005]. LS may be viewed as a very particular case of GA whose population is composed of only one combination. Algorithm 2 describes this basic principle, the main steps of which are briefly described in the next paragraphs.

Algorithm 2: Local Search (LS)

Generate an initial combination $s \in S$
while *stopping criteria not reached* **do**
  Choose $s' \in n(s)$
  $s \leftarrow s'$
return *the best combination encountered during the search*

Neighbourhood function: The LS algorithm is parameterised by a neighbourhood function $n : S \rightarrow \mathscr{P}(S)$ which defines the set of combinations $n(s)$ that may be obtained by applying some transformation operators to a given combination $s \in S$. One may consider different kinds of transformation operators such as, for example, changing the value of one variable, or swapping the values of two variables. Each different transformation operator induces a different neighbourhood, the size of which may vary. Hence, the choice of the transformation operators has a strong influence on the solution process. A strongly desirable property of the transformation operator is that it must allow the search to reach the optimal combination from any initial combination. In other words, the directed graph which associates a vertex with each combination of S, and an edge (s_i, s_j) with each couple of combinations such that $s_j \in n(s_i)$, must contain a path from any of its vertices to the vertex associated with the optimal combination.

Exemple 2 *For the TSP, the* 2-opt *operator consists in deleting deux edges, and replacing them by two new edges that reconnect the two paths created by the edge deletion. More generally, the* k-opt *operator consists in deleting k mutually disjoint edges and re-assembling the different sub-paths created by these deletions by adding k new edges in such a way that a complete tour is reconstituted. The larger k, the larger the neighbourhood size.*

Generation of the initial combination: The search is started from a combination which is often randomly generated. The initial combination may also be generated with a greedy approach such as those introduced in 3.1. When local search is hybridised with another meta-heuristic such as, for example, genetic algorithms or ant colony optimisation, the initial combination may be the result of another search process.

Choice of a neighbour: At each iteration of LS, one has to choose a combination s' in the neighbourhood of the current combination s and substitute s with s' (this is called a *move*). There exist many different strategies for choosing a neighbour. For example, *greedy strategies* always choose better (or at least as good) neighbours. In particular, the *best improvement* greedy strategy scans the whole neighbourhood and selects the best neighbour, that is, the one which most improves the objective function [Selman et al, 1992], whereas the *first improvement* greedy strategy selects the first neighbour which improves the objective function. These greedy strategies may be compared to *hill climbers* that always choose a raising path. This kind of strategy usually allows the search to quickly improve the initial combination. However, once the search has reached a locally optimal combination —that is, a combination

whose neighbours all have worse objective function values— it becomes stuck on it. To escape from these local optima, one may consider different alternative strategies offered by meta-heuristics like:

- *random walk* [Selman et al, 1994], that allows with a very small probability (controlled by a parameter) to randomly select a neighbour;
- *simulated annealing* [Aarts and Korst, 1989], that allows to select neighbours of worse quality with respect to a probability that decreases with time;
- *tabu search* [Glover and Laguna, 1993], that prevents the search from cycling on a small subset of combinations around local optima by memorising the last moves in a tabu list, and forbidding inverse moves to these tabu moves.

Repetition of local search: Local search may be repeated several times, starting from different initial combinations. These initial combinations may be randomly and independently generated, as proposed in *multi-start local search*. They may also be obtained by perturbing a combination generated during the previous local search process, as proposed in *iterated local search* [Lourenco et al, 2002] and *breakout local search* [Benlic and Hao, 2013a,b]. We may also perform several local searches in parallel, starting from different initial combinations, and evenly redistributing current combinations by removing the worst ones and duplicating the best ones, as proposed in *go with the winner* [Aldous and Vazirani, 1994].

Local search with multiple neighbourhoods: A typical local search algorithm usually relies on a single neighbourhood to explore the search space. However, in a number of settings, several neighbourhoods can be jointly employed to reinforce the search capacity of local search. Variable neighborhood search is a well-known example, which employs the greedy strategy and a strict neighbourhood transition rule to examine a set of nested neighbourhoods with increasing sizes. Each time a local optimum is reached within the current neighborhood, the search switches to the next (larger) neighborhood and switches back again to the smallest neighourhood once an improving solution is found [Hansen and Mladenovic, 2001]. Other local search algorithms using more flexible neighbourhood transition rules can be found in [Goëffon et al, 2008; Ma and Hao, 2017; Wu et al, 2012].

3 Constructive Meta-heuristics

Constructive approaches build one or more combinations in an incremental way: starting from an empty combination, they iteratively add combination components until obtaining a complete combination. These approaches are said to be *model-based* in [Zlochin et al, 2004], as they use a model (which is often stochastic) to choose, at each iteration, the next component to be added to the partial combination.

There exist different strategies to choose the components to be added, at each iteration, the most well known being greedy randomised strategies, described in Section 3.1, Estimation of Distribution Algorithms, described in Section 3.2, and Ant Colony Optimisation, described in Section 3.3.

3.1 Greedy Randomised Algorithms

A greedy algorithm builds a combination in an incremental way: it starts from an empty combination and incrementally completes it by adding components to it. At each step, the component to be added is chosen in a greedy way, that is, one chooses the component which maximises some problem-dependent heuristic function which locally evaluates the interest of adding the component with respect to the objective function. A greedy algorithm usually has a very low time complexity, as it never backtracks to a previous choice. The quality of the final combination depends on the heuristic.

Exemple 3 *A greedy algorithm for the TSP may be defined as follows: starting from an initial city which is randomly chosen, one chooses, at each iteration, the closest city that has not yet been visited, until all cities have been visited.*

Greedy randomised algorithms deliberately introduce a slight amount of randomness into greedy algorithms in order to diversify the constructed combinations. In this case, greedy randomised constructions are iteratively performed, until some stopping criteria is reached, and the best constructed combination is returned. To introduce randomness in the construction, one may randomly choose the next component within the k best ones, or within the set of components whose quality is bounded by a given ratio with respect to the best component [Feo and Resende, 1989]. Another possibility is to select the next component with respect to probabilities which are defined proportionally to component qualities [Jagota and Sanchis, 2001].

Exemple 4 *For the TSP, if the last visited city is i, and if C contains the set of cities that have not yet been visited, we may define the probability to select a city $j \in C$ by $p(j) = \frac{[1/d(i,j)]^\beta}{\sum_{k \in C}[1/d(i,k)]^\beta}$. β is a parameter that allows one to tune the level of greediness/randomisation: if $\beta = 0$, then all cities in C have the same probability to be selected; the higher β, the higher the probability of selecting cities that are close to i.*

3.2 Estimation of Distribution Algorithms

Estimation of Distribution Algorithms (EDAs) are greedy randomised algorithms [Larranaga and Lozano, 2001]: at each iteration, a set of combinations is generated according to a greedy randomised principle as described in the previous section. However, EDAs take benefit of previously computed combinations to bias the construction of new combinations. Algorithm 3 describes this basic principle, the main steps of which are briefly described in the next paragraphs.

Generation of the initial population: In most cases, the initial population is randomly generated with respect to a uniform distribution, and only the best constructed combinations are kept in the population.

Algorithm 3: Estimation of Distribution Algorithm (EDA)

```
begin
    Generate an initial population of combinations P ⊆ S
    while stopping criteria not reached do
        Use P to construct a probabilistic model M
        Use M to generate new combinations
        Update P with respect to these new combinations
    return the best combination that has been built during the search process
```

Construction of a probabilistic model: Different kinds of probabilistic models may be considered. The simplest one, called PBIL [Baluja, 1994], is based on the probability distribution of each combination component, independently from other components. In this case, one computes for each component its occurrence frequency in the population, and one defines the probability of selecting this component proportionnaly to its frequency. Other models may take into account dependency relationships between components by using bayesian networks [Pelikan et al, 1999]. In this case, the dependency relationships between components are modelled by edges in a graph, and conditional probability distributions are associated with these edges. Such models usually allow the search to build better combinations, but they are also more expensive to compute.

Exemple 5 *For the TSP, if the last visited city is i, and if C contains the set of cities that have not yet been visited, we may define the probability to select a city* $j \in C$ *by* $p(j) = \frac{freq_P(i,j)}{\sum_{k \in C} freq_P(i,k)}$*, where* $freq_P(i,j)$ *is the number of combinations of P that use edge* (i,j)*. Hence, the more the population uses edge* (i,j)*, the higher the probability to select* j*.*

Generation of new combinations: New combinations are built in a greedy randomised way, using the probabilistic model to choose components.

Update of the population: In most cases, only the best combinations are kept in the population for the next iteration of the search process, whatever they belong to the current population or to the set of new generated combinations. However, it is also possible to keep lower quality combinations in order to maintain a good diversity in the population, thus ensuring a good sampling of the search space.

3.3 Ant Colony Optimisation

There exists a strong similarity between Ant Colony Optimisation (ACO) and EDAs [Zlochin et al, 2004]. Both approaches use a probabilistic model to build new combinations, and in both approaches, this probabilistic model evolves during the search process with respect to previously built combinations, in an iterative learning process. The originality and the main contribution of ACO are that it borrows features from the collective behaviour of ants to update the probabilistic model. Indeed, the

probability of choosing a component depends on a quantity of pheromone which represents the past experience of the colony with respect to the choice of this component. This quantity of pheromone evolves by combining two mechanisms. The first mechanism is a pheromone laying step: pheromone trails associated with the best combinations are reinforced in order to increase the probability of selecting these components. The second mechanism is pheromone evaporation: pheromone trails are uniformly and progressively decreased in order to progressively forget older experience. Algorithm 4 describes this basic principle, the main steps of which are briefly described in the next paragraphs.

Algorithm 4: Ant Colony Optimisation

```
Initialise pheromone trails to τ0
while Stopping conditions are not reached do
    Each ant builds a combination
    Pheromone trails are updated
return the best combination
```

Pheromone trails: Pheromone is used to bias selection probabilities when building combinations in a greedy randomised way. A key point lies in the choice of the components on which pheromone is laid. At the beginning of the search, all pheromone trails are initialised to a given value τ_0.

Exemple 6 *For the TSP, a trail τ_{ij} is associated with each pair of cities (i, j). This trail represents the past experience of the colony with respect to visiting i and j consecutively.*

Construction of a combination by an ant: At each cycle of an ACO algorithm, each ant builds a combination according to a greedy randomised principle, as introduced in 3.1. Starting from an empty combination (or a combination that contains a first combination component), at each iteration, the ant selects a new component to be added to the combination, until the combination is complete. At each iteration, the next combination component is selected with respect to a probabilistic transition rule: given a partial combination X, and a set C of combination components that may be added to X, the ant selects a component $i \in C$ with probability:

$$p_X(i) = \frac{[\tau_X(i)]^\alpha \cdot [\eta_X(i)]^\beta}{\sum_{j \in C}[\tau_X(j)]^\alpha \cdot [\eta_X(j)]^\beta} \quad (1)$$

where $\tau_X(i)$ (resp. $\eta_X(i)$) is the pheromone (resp. heuristic) factor associated with component i, given the partial combination X (the definition of this factor is problem-dependant), and α and β are two parameters used to balance the relative influence of pheromone and heuristic factors in the transition probability. In particular, if $\alpha = 0$ then the pheromone factor does not influence the selection and the algorithm behaves like a pure greedy randomised algorithms. On the contrary, if $\beta = 0$ then transition probabilities only depend on pheromone trails.

Exemple 7 *For the TSP, the pheromone factor $\tau_X(i)$ is defined by the quantity of pheromone τ_{ki} laying between the last city k visited in X and the candidate city i. The heuristic factor is proportionally inverse to the distance between the last city visited in X and the candidate city i.*

Pheromone updating step: Once each ant has built a combination, pheromone trails are updated. First, they are decreased by multiplying each trail with a factor $(1-\rho)$, where $\rho \in [0;1]$ is the evaporation rate. Then, some combinations are "rewarded", by laying pheromone on their components. There exist different strategies for selecting the combinations to be rewarded. We may reward all combinations built during the current cycle, or only the best combinations of the current cycles, or the best combination found since the beginning of the search. These different strategies influence the search intensification and diversification. In general, the quantity of pheromone added is proportional to the quality of the rewarded combination. This quantity is added on the pheromone trails associated with the rewarded combination. To prevent the algorithm from premature convergence, pheromone trails may be bounded between two bounds, τ_{min} and τ_{max}, as proposed in the *MAX-MIN Ant System* [Stützle and Hoos, 2000].

Exemple 8 *For example, for the TSP, we add pheromone on each trail τ_{ij} such that cities i and j have been consecutively visited in the rewarded combination.*

4 Hybrid Meta-heuristics

The different meta-heuristics presented in Sections 2 and 3 may be combined to define new meta-heuristics. Two classical examples of such hybridisations are described in Sections 4.1 and 4.2.

4.1 Memetic Algorithms

Memetic algorithms combine population-based approaches (evolutionary algorithms) with local search [Moscato, 1999; Neri et al, 2012]. This hybridisation aims at taking benefit of the diversification abilities of population-based approaches and intensification abilities of local search [Hao, 2012].

A memetic algorithm may be viewed as a genetic algorithm (as described in Algorithm 1) extended by a local search process. As in genetic algorithms, a population of combinations is used to sample the search space, and a recombination operator is used to create new combinations from combinations of the population. Selection and replacement mechanisms are used to determine the combinations that are recombined and those that are eliminated. However, the mutation operator of genetic algorithms is replaced by a local search process, that may be viewed as a guided macro-mutation process. The goal of the local search step is to improve

the quality of the generated combinations. It mainly aims at intensifying the search by exploiting search paths determined by the considered neighbourhood operators. Like recombination, local search is a key component of memetic approaches.

4.2 Hybridisation between Perturbative and Constructive Approaches

Perturbative and constructive approaches may be hybridised in a straightforward way: at each iteration, one or more combinations are built according to a greedy randomised principle (that may be guided by EDA or ACO), and then some of these combinations are improved with a local search process. This hybridisation is called GRASP (*Greedy Randomised Adaptive Search Procedure*) [Resende and Ribeiro, 2003]. A key point is the choice of the neighbourhood operator and the strategy used to select moves for the local search. The goal is to find a compromise between the time spent by the local search to improve combinations, and the quality of these improvements. Typically, one chooses a simple greedy local search, that improves combinations until reaching a local optimum.

Note that the best performing EDA and ACO algorithms usually include this kind of hybridisation with local search.

5 Intensification *versus* Diversification

For all meta-heuristics described in this chapter, a key point highly relies on their capability to find a suitable balance between two dual goals:

- *intensification* (also called *exploitation*), which aims at guiding the search towards the most promising areas of the search space, that is, around the best combinations found so far;
- *diversification* (also called *exploration*), which aims at allowing the search to move away from local optima and discover new areas, that may contain better combinations.

The way the search is intensified/diversified depends on the considered meta-heuristic and is achieved by modifying parameters. For perturbative approaches, intensification is achieved by favouring the best neighbours: for genetic algorithms, elitist selection and replacement strategies favour the reproduction of the best combinations; for local search, greedy strategies favour the selection of the best neighbours. Diversification of perturbative approaches is usually achieved by introducing a mild amount of randomness: for genetic algorithms, diversification is mainly achieved by mutation; for local search, diversification is achieved by allowing the search to select ”bad” neighbours with small probabilities.

For constructive approaches, intensification is achieved by favouring, at each step of the construction, the selection of components that belong to the best combinations built so far. Diversification is achieved by introducing randomness in the selection procedures, thus allowing to choose (with small probabilities) worse components.

In general, the more the search is intensified, the quicker it converges towards rather good combinations. However, if one over-intensifies search, the algorithm may stagnate around local optima, concentrating all the search effort on a small sub-area without being able to escape from this attracting sub-area to explore other areas. On the contrary, if one over-diversifies the search, so that it behaves like a random search, the algorithm may spend most of its time on exploring poor quality combinations. It is worth mentioning here that the *right* balance between intensification and diversification clearly depends on the CPU time the user is willing to spend on the solution process: the shorter the CPU time limit, the more the search should be intensified to quickly converge towards solutions. The right balance between intensification and diversification also highly depends on the instance to be solved or, more precisely, on the topology of its search landscape. In particular, if there is a good correlation between the quality of a locally optimal combination and its distance to the closest optimal combination (such as *massif central* landscapes), then the best results are usually obtained with a strong intensification of the search. On the contrary, if the search landscape contains a lot of local optima which are uniformly distributed in the search space independently from their quality, then the best results are usually obtained with a strong diversification of the search.

Different approaches have proposed to dynamically adapt parameters that balance intensification and diversification during the search process. This is usually referred to as *reactive search* [Battiti et al, 2008]. For example, the reactive tabu search approach proposed in [Battiti and Protasi, 2001] dynamically adapts the length of the tabu list by increasing it when combinations are recomputed (thus indicating that it is turning around a local optima), and decreasing it when the search has not recomputed combinations for a while. Also, the *IDwalk* local search of [Neveu et al, 2004] dynamically adapts the number of neighbours considered at each move. More generally, machine learning techniques may be used to automatically learn how to dynamically adapt parameters during the search [Battiti and Brunato, 2017].

Other approaches have studied how to automatically search for the best static parameter settings, either for a given class of instances (parameter configuration), or for each new instance to solve (parameter selection) [Hoos et al, 2017].

6 Applications in Artificial Intelligence

Meta-heuristics have been applied successfully to solve a very large number of classical NP-hard problems and practical applications in highly varied areas. In this section, we discuss their application to two central problems in artificial intelligence: satisfiability of Boolean formulas (SAT) and satisfaction of constraints (CSP).

6.1 Satisfiability of Boolean Formulas

SAT is one of the central problems in artificial intelligence. For a Boolean formula, SAT involves determining a model, namely an assignment of a Boolean value (true or false) to each variable such that the valuation of the formula is true. For practical reasons, it is often assumed that the Boolean formula under consideration is given in its clausal form (CNF) even if from a general point of view, this is not a necessary condition to apply a meta-heuristic.

Note that SAT per se is not an optimisation problem and does not have an explicit objective function to optimise. Since meta-heuristics are generally conceived to solve optimisation problems, they consider a more general problem MAXSAT whose goal is to find the valuation maximising the number of satisfied clauses. In this context, the result of a meta-heuristic algorithm to an instance MAXSAT can be of two kinds: either the returned assignment satisfies all the clauses of the formula, in which case a solution (model) is found for the given SAT instance, or it does not satisfy all the clauses, in which case we can not know if the given instance is satisfiable or not.

The search space of a SAT instance is naturally given by the set of possible assignments of Boolean values to the set of variables. Thus, for an instance with n variables, the size of the search space is 2^n. The objective function (also called evaluation function) counts the number of satisfied clauses. This function introduces a total order on the combinations of the search space. We can also consider the dual objective (or evaluation) function, counting the number of falsified clauses, and corresponding to a penalty function to minimise: each falsified clause has a penalty weight equal to 1, and a combination with an evaluation of 0 indicates a solution (model). This function can be further fine-tuned by a dynamic penalty mechanism or an extension including other information than the number of falsified clauses.

A lot of work has been done during the last decades for practical solving of SAT. Various competitions on SAT solvers regularly organised by the scientific community continually boost research activities, assessment and comparison of SAT algorithms. These researches resulted in a very large number of contributions that improve the performance and robustness of SAT algorithms, especially stochastic local search (SLS).

6.1.1 Stochastic Local Search

SLS algorithms generally consider the following simple neighbourhood function: two combinations are neighbouring if their Hamming distance is exactly 1. The transition from one combination to another is conveniently achieved by flipping a variable. For a formula with n variables, the neighbourhood has a size of n (since each of the n variable can be flipped to obtain a neighbour). This neighbourhood can be shrunk by limiting the choice of the modified variable to those that appear in at least one falsified clause. This reduced neighbourhood is often used in SLS

algorithms, because in addition to its reduced size, it makes the search more focused (recall that the goal is to satisfy all the clauses).

SLS algorithms differ essentially in the techniques used to find the suitable compromise between 1) exploration of the search space sufficiently broad to reduce the risk of search stagnation in a non-promising region and 2) exploitation of available information to discover a solution in the region currently under examination. In order to classify these different algorithms, we can consider on the one hand the way the constraints are exploited, and on the other hand the way the search history is used [Bailleux and Hao, 2008].

Exploitation of instance structures

The structure of the given problem instance is induced by its clauses. We can distinguish three levels of exploitation of this structure.

The search can be simply guided by the objective function, so that only the number of clauses falsified by the current assignment is taken into account. A typical example is the popular GSAT algorithm [Selman et al, 1992] that, at each iteration, randomly selects a neighbouring assignment from those minimising the number of falsified clauses. This greedy descent strategy is an aggressive exploitation technique that can be easily trapped in local optima. To work around this problem, GSAT uses a simple diversification technique based on the restart of the search from a new initial configuration after a fixed number of iterations. Several variants of GSAT such as CSAT et TSAT [Gent and Walsh, 1993] and simulated annealing [Spears, 1996] are based on the same principle of minimising the number of falsified clauses.

The search can also be guided by conflicts, so that falsified clauses are explicitly taken into account. This approach is particularly useful when the neighbourhood no longer contains a combination improving the objective value. In order to better guide the search, other information obtained via, for example, an analysis of falsified clauses can be used. The archetype example of this type of algorithms is WALKSAT [McAllester et al, 1997] that, at each iteration, randomly selects one of the falsified clauses in the current assignment. A heuristic is then used to choose one of the variables of this clause, the value of which will be modified to obtain the new assignment. The GWSAT algorithm (GSAT with random walk) [Selman and Kautz, 1994] also uses a random walk guided by falsified clauses that, at each iteration, modifies a variable belonging to a falsified clause with a probability p (called noise) and uses the descent strategy of GSAT with a probability $1-p$.

Finally, *exploitation of deductive constraints* makes it possible to use deductive rules to modify the current combination. This is the case for UNITWALK [Hirsch, 2005], which uses the unit resolution to determine the value of some variables of the current assignment from the values of other variables fixed by local search. Another type of deductive approach is used in NON-CNF ADAPT NOVELTY [Pham et al, 2007], which analyses the formula to be processed to search for dependency links between variables.

Exploitation of search history

To classify SLS approaches for SAT, we can also consider the way the action history is exploited since the beginning of the search as well as possibly their effects. There are three levels of exploitation of this history.

For *Markov algorithms (or without memory)*, the choice of each new combination depends only on the current combination. The algorithms GSAT, GRSAT, WALKSAT/SKC, as well as the general simulated annealing algorithm are typical examples of SLS algorithms without memory. These algorithms have the advantage of minimising the processing time necessary for each iteration, yet they have been outperformed in practice during the last decades by algorithms with memory.

For *algorithms with short-term memory*, the choice of each new combination takes into account a history of all or part of the changes of the variable values. SAT solvers based on tabu search, such as WALKSAT/TABU [McAllester et al, 1997] or TSAT [Mazure et al, 1997] are typical examples. Some algorithms like WALKSAT, NOVELTY, RNOVELY, and G2WSAT [Li and Huang, 2005] also integrate *aging information* in the choice criterion. Typically, this criterion is used to favour amongst several candidate variables the oldest modified one.

For *algorithms with long-term memory*, the choice of each new combination depends on choices made since the start of the search and their effects, in particular in terms of satisfied and falsified clauses. The idea is to use a learning mechanism to take advantage of the failures and accelerate the search. In practice, the history of falsified clauses is exploited by associating weights with clauses, the objective being to diversify the search by forcing it to take new directions. A first example is weighted-GSAT [Selman and Kautz, 1993] where, after each change of a variable, weights of falsified clauses are incremented. The score used by the search process is simply the sum of the weights of the falsified clauses. As the weight of frequently falsified clauses increases, the process tends to favour the modification of variables belonging to these clauses, until the evolution of the weights of the other clauses influences the search again. Other examples include DLM (*discrete lagrangian method*) [Shang and Wah, 1998], DDWF (*Divide and Distribute Fixed Weight*) [Ishtaiwi et al, 2005], PAWS (*Pure Additive Weighting Scheme*) [Thornton et al, 2004], ESG (*Exponentiated Subgradient Algorithm*) [Schuurmans et al, 2001], SAPS (*Scaling and Probabilistic Smoothing*) [Hutter et al, 2002] and WV (*weighted variables*) [Prestwich, 2005].

6.1.2 Population-based Algorithms

Genetic algorithms (GAs) have been repeatedly applied to SAT [Jong and Spears, 1989; Young and Reel, 1990; Crawford and Auton, 1993; Hao and Dorne, 1994]. Like local search algorithms, these GAs adopt a representation of a candidate combination as an assignment of values 0/1 to the set of variables. First GAs handle general formulas, not limited to the CNF form. Unfortunately, the results obtained

by these algorithms are generally disappointing when they are applied to SAT benchmarks.

The first GA dealing with CNF formulas is presented in [Fleurent and Ferland, 1996]. The originality of this algorithm is the use of a specific and original cross-over operator that tries to exploit the semantics of two parent assignments. Given two parents p_1 and p_2, one examines the clauses that are satisfied by one parent, but falsified by the other parent. When a clause is true in one parent p_1 and false in the other parent, the values of all variables in this clause are passed directly to the child. The memetic algorithm integrating this cross-over operator and a tabu search procedure yielded very interesting results during the second DIMACS implementation challenge.

Another representative hybrid genetic algorithm is GASAT [Lardeux et al, 2006]. As the algorithm of [Fleurent and Ferland, 1996], GASAT attaches a preponderant importance to the design of a semantic cross-over operator. Thus, a new class of cross-overs is introduced aimed at correcting the "errors" of the parents and combining their good characteristics. For example, if a clause is false in two good-quality parents, we can force this clause to become true in the child by flipping a variable of the clause. The intuitive argument that justifies this consists in considering such a clause as difficult to satisfy otherwise and consequently it is preferable to satisfy the clause by force. Similarly, recombination mechanisms are developed to exploit the semantics associated with a clause that is made simultaneously true by both parents. With this type of cross-overs and a tabu search algorithm, GASAT is very competing on some categories of SAT benchmarks.

FlipGA [Rossi et al, 1999] is a genetic algorithm that relies on a standard uniform cross-over that generates, by an equiprobable mixture of the values of both parents, a child which is further improved by local search.

Other GAs are presented in [Eiben and van der Hauw, 1997; Gottlieb and Voss, 1998; Rossi et al, 2000], but they are in fact local search algorithms since the population is reduced to a single assignment. Their interest lies in the techniques used to refine the basic evaluation function (the number of falsified clauses) by a dynamic adjustment during the search process. We find in [Gottlieb et al, 2002] an experimental comparison of these few genetic algorithms for SAT. However, their practical interest remains to be demonstrated since they have rarely been directly confronted with state-of-the-art SLS algorithms.

Note that in these population-based algorithms, local search is often an essential component in order to reinforce intensification capabilities. The cross-over's role may differ depending on whether it is completely random [Rossi et al, 1999] in which case it is used essentially to diversify the search, or is based on the semantics of the problem [Fleurent and Ferland, 1996; Lardeux et al, 2006] in which case it allows both to diversify and intensify the search.

6.2 Constraint Satisfaction Problems

A constraint satisfaction problem (CSP) is a combinatorial problem modelled in the form of a set of constraints defined over a set of variables, each of these variables taking its values in a given domain.

As for SAT, one generally considers the optimisation problem MaxCSP whose objective is to find a complete assignment (assigning a domain value to each variable) that maximises the number of satisfied constraints. The search space is thus defined by the set of all possible complete assignments, while the objective function to be maximised counts the number of satisfied constraints for a complete assignment.

6.2.1 Genetic Algorithms

In its simplest form, a genetic algorithm for CSPs uses a population of complete assignments that are recombined by simple cross-over, as well as mutation consisting of changing the value of a variable. An experimental comparison of eleven genetic algorithms for solving binary CSPs is presented in [Craenen et al, 2003]. It is showed that the three best algorithms (*Heuristics GA version 3*, *Stepwise Adaptation of Weights* et *Glass-Box*) have equivalent performances and are significantly better than the other eight algorithms. However, these best genetic algorithms are clearly not competitive, either with exact approaches based on a tree search, or with other heuristic approaches, such as local search or Ant colony optimisation [van Hemert and Solnon, 2004].

Other genetic algorithms have been proposed for particular CSPs. These specific algorithms exploit knowledge of the constraints of the problem to be solved in order to define better cross-over and mutation operators, leading to better results. We can cite in particular [Zinflou et al, 2007] that obtains competitive results for a car sequencing problem.

6.2.2 Local Search

There is a great deal of work on CSP solving using local search techniques, and this approach generally yields excellent results. These local search algorithms for CSPs differ essentially by the neighbourhood and the selection strategy considered.

Neighbourhood. Given a candidate assignment, a move operation typically modifies the value of a variable. So the neighbourhood of an assignment is composed of all assignments that can be obtained by changing the value of a variable in this assignment. Depending on the nature of the constraints, it is possible to consider other neighbourhoods, such as the neighbourhood induced by swap moves (that exchange the values of two variables), typically considered when there is a permutation constraint (that enforces a given set of variables to be assigned to a permutation of a given set of values).

Some studies consider neighbourhoods not only between complete assignments, but also between partial assignments. In particular, the *decision repair* approach in [Jussien and Lhomme, 2002] combines filtering techniques such as those described in Chapter 6 of this volume with local search on the space of partial assignments. Given a current partial assignment A, if the filtering detects an inconsistency then the neighbourhood of A is the set of assignments resulting from the removal of a "variable/value" couple of A, otherwise the neighbourhood of A is the set of assignments resulting from the addition of a "variable/value" couple to A.

Selection strategies. Many strategies for choosing the next move have been proposed. The *min-conflict* strategy [Minton et al, 1992] randomly selects a variable involved in at least one violated constraint and chooses for this variable the value that minimises the number of constraint violations. This greedy strategy, which is famous for having found solutions to the N-queen problem for a million of queens, tends to be easily trapped in *local optima*. A simple and classical way to get the *min-conflict* strategy out of local minima is to combine it with the random walk strategy [Wallace, 1996]. Other strategies for choosing the variable/value pair are studied in [Hao and Dorne, 1996].

Local search using tabu search (TabuCSP) [Galinier and Hao, 1997] obtains excellent results for binary CSPs. Starting from an initial assignment, the idea is to choose the non-tabu move that most increases the number of satisfied constraints at each iteration. A move consists in changing the value of a *conflicting variable* (i.e., a variable involved in at least one unsatisfied constraint). Each time a move is performed, it is forbidden to select the move again during the next k iterations (the move is said to be tabued, k being the tabu tenure). However, a move leading to a solution better than any discovered solution is always performed even if the move is currently declared as tabu. This selection criterion is called *aspiration* in the terminology of tabu search.

CBLS (Constraint Based Local Search) [Hentenryck and Michel, 2005] adapts ideas of constraint programming to local search and allows one to quickly design local search algorithms for solving CSPs. In particular, it introduces the concept of incremental variable allowing an incremental evaluation of neighbourhoods, and uses invariants that are stated declaratively to achieve this. In [Björdal et al, 2015], a CBLS backend is described for the MiniZinc CSP modelling language. Other generic systems of CSP solving based on local search are presented in [Davenport et al, 1994; Nonobe and Ibaraki, 1998; Codognet and Diaz, 2001; Galinier and Hao, 2004].

6.2.3 Greedy Construction Algorithms

We can construct a complete assignment for a CSP according to the following greedy principle: starting from an empty assignment, we iteratively select an unassigned variable and a value for this variable, until all variables receive a value. To choose a variable and a value at each step, we can use the ordering heuristics that are employed by exact approaches described in Chapter 6 of this volume. A well-

known example of using this principle is the DSATUR algorithm [Brélaz, 1979] for the graph coloring problem, which is a particular CSP. DSATUR constructs a coloring by choosing at each iteration the uncoloured vertex having the largest number of neighbours colored with different colors (the most saturated neighbour). Ties are broken by choosing the vertex of the highest degree. The selected vertex is then colored by the smallest possible color.

6.2.4 Ant Colony Optimisation

Ant colony optimisation has been applied to CSPs in [Solnon, 2002, 2010]. The idea is to construct complete assignments according to a random greedy principle: starting from an empty assignment, one selects at each iteration an unassigned variable and a value to be assigned to this variable, until all variables are assigned. The main contribution of ACO is to provide a value selection heuristic: it is chosen according to a probability that depends on a heuristic factor (inversely proportional to the number of new violations introduced by the value), and a pheromone factor that reflects the past experience with the use of this value.

The pheromone structure is generic and can be used to solve any CSP. It associates a pheromone trail to each variable x_i and each value v_i that can be assigned to x_i: intuitively, this trail represents the colony's past experience of assigning value v_i to variable x_i. Other pheromone structures have been proposed for solving particular CSPs such as the car sequencing problem [Solnon, 2008] or the multidimensional knapsack problem [Alaya et al, 2007].

Ant colony optimisation has been integrated into general CSP libraries and IBM/Ilog Solver for constraint optimisation problems and CP Optimizer [Khichane et al, 2008, 2010]. These generic systems make it possible to use high level languages to describe in a declarative way the problem to be solved, the solution of the problem is automatically supported by an ACO algorithm built into the language.

7 Discussions

Meta-heuristics have been used successfully to solve many difficult combinatorial search problems, and these approaches often obtain very good results during various implementation competitions, either for solving real problems such as car sequencing, timetabling and nurse rostering, or well-known NP-hard problems such as the SAT problem and many graph problems (e.g., graph coloring).

The variability of meta-heuristics naturally raises the question of how to choose the most suitable meta-heuristic to solve a given problem. Obviously, this question is complex and comes close to a fundamental quest in artificial intelligence, namely the automatic problem solving. We are only discussing some of the answers.

In particular, the choice of a meta-heuristic depends on the relevance of its basic mechanisms for the problem considered, that is, their ability to generate good combinations:

- For GAs, the recombination operator should be able to identify interesting patterns that can be assembled and recombined [Hao, 2012]. For example, for graph coloring, an interesting pattern is a group of vertices of the same color and shared by good solutions; This type of information allowed the design of very successful cross-over operators with two parents [Dorne and Hao, 1998; Galinier and Hao, 1999] or with several parents [Galinier et al, 2008; Malaguti et al, 2008; Lü and Hao, 2010; Porumbel et al, 2010].
- For local search, the neighbourhood must favour the construction of better combinations. For example, for the SAT and CSP problems, the neighbourhood centred on conflicting variables (see section 6.2) is relevant because it favours elimination of conflicts.
- For ACO, the pheromone structure must be able to guide the search for better combinations. For example, for CSP, the pheromone structure associating a pheromone trail with each variable/value couple is relevant because it allows to learn what are the right values to be assigned to each variable.

Another crucial point is the time complexity of the elementary operators of the meta-heuristic considered (recombination and mutation for GAs, move for local search, addition of a solution component for constructive approaches, ...). This complexity depends on the data structures used to represent the combinations. Thus, the choice of a meta-heuristic depends on the existence of data structures that allow a fast evaluation of the evaluation function after each application of the elementary operators of the meta-heuristic.

Other important elements are shared by all meta-heuristics. In particular, the evaluation function (which may be the same as or different from the objective function of the problem) is a key element because it measures the relevance of a combination. For example, for CSP, the evaluation function can simply count the number of unsatisfied constraints, but in this case the relative importance of each constraint violation is not recognized. An interesting refinement is to introduce into the evaluation function a penalty term to quantify the degree of violation of a constraint [Galinier and Hao, 2004]. This function can be further refined, as in the case of the SAT problem (see section 6.1), by using weights that can be dynamically adjusted according to the history of violations of each constraint.

Meta-heuristic algorithms often have a number of parameters that have a significant influence on their efficiency. Thus, we can consider the development of a meta-heuristic algorithm as a configuration problem: the goal is to choose the best building blocks to combine (recombination or mutation operators, neighbourhoods, move selection strategies, heuristic factors, etc.) as well as the best parameter setting (mutation rate, evaporation rate, noise, tabu tenure, the weight of pheromone structures and heuristic factors, etc.). A promising approach to solve this configuration problem consists in using automatic configuration algorithms to design the

algorithm which is best suited to a set of given instances [Bezerra et al, 2016; Hutter et al, 2017; Xu et al, 2010].

We terminate this chapter with a word of caution. In the last years, the community has witnessed the appearance of a great number of metaphor-based or nature-inspired metaheuristics. These ”novel” methods are often proposed by recasting a natural phenomenon or species in terms of a search method without a real justification or understanding of the underlying search strategies. The proliferation of these fancy methods pollutes in some sens the research in the area of metaheuristics and makes it difficult for people to figure out which are the true methods that can be used. Fortunately, the dangers related to this proliferation begin to be recognized by the research community, as justly analyzed in [Sörensen, 2013]. On the other hand, other interesting trends begin to emerge and attract the attention of the meta-heuristic community. In the continuation of the effort of creating hybridised methods, the combination of artificial intelligence techniques (e.g., machine learning and data mining) and meta-heuristics seems to offer a great potential for creating improved search methods, as exemplified by the studies reported in [] [Je vais ajouter quelques references recentes ici.].

8 Conclusion

In this chapter, we have presented a panorama of meta-heuristics, a class of general methods useful to solve difficult combinatorial search problems. Even if these methods have no provable guarantees on the distance of the achieved solution to the optimum of the given problem, they have the advantage of being virtually applicable to any difficult search problem. Meanwhile, to obtain an effective search algorithm, it is critical to adapt the general search strategies offered by these general methods to the problem at hand. In particular, the targeted problem needs to be understood in depth to identify problem specific knowledge, which can be then incorporated into the search components of the algorithm. The pursuit design goal is to build an algorithm that is able to ensure a balanced exploitation and exploration of the search space. It is equally important to apply the lean design principle in order to avoid redundant or superficial algorithmic components. To sum up, meta-heuristic framework constitutes an interesting enrichment to the arsenal of existing search methods and offers a valuable alternative for tackling hard combinatorial problems.

References

Aarts EH, Korst JH (1989) Simulated annealing and Boltzmann machines: a stochastic approach to combinatorial optimization and neural computing. John Wiley & Sons, Chichester, U.K.

Alaya I, Solnon C, Ghedira K (2007) Optimisation par colonies de fourmis pour le probléme du sac-é-dos multi-dimensionnel. Techniques et Sciences Informatiques (TSI) 26(3-4):271–390

Aldous D, Vazirani UV (1994) “go with the winners” algorithms. In: 35th Annual Symposium on Foundations of Computer Science, pp 492–501

Bailleux O, Hao JK (2008) Algorithmes de recherche stochastiques. In: Saïs L (ed) Problème SAT : progrès et défis, Hermès - Lavoisier, chap 5

Baluja S (1994) Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Tech. rep., Carnegie Mellon University, Pittsburgh, PA, USA

Battiti R, Brunato M (2017) The LION way. Machine Learning *plus* Intelligent Optimization. LIONlab, University of Trento, Italy, URL `http://intelligent-optimization.org/LIONbook/`

Battiti R, Protasi M (2001) Reactive local search for the maximum clique problem. Algorithmica 29(4):610–637

Battiti R, Brunato M, Mascia F (2008) Reactive Search and Intelligent Optimization. Springer Verlag

Benlic U, Hao J (2013a) Breakout local search for the quadratic assignment problem. Applied Mathematics and Computation 219(9):4800–4815

Benlic U, Hao J (2013b) Breakout local search for the vertex separator problem. In: IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013, pp 461–467

Bezerra LCT, López-Ibáñez M, Stützle T (2016) Automatic component-wise design of multiobjective evolutionary algorithms. IEEE Transactions on Evolutionary Computation 20(3):403–417

Björdal G, Monette JN, Flener P, Pearson J (2015) A constraint-based local search backend for minizinc. Constraints 20:325–345

Brélaz D (1979) New methods to color the vertices of a graph. Journal of the CACM 22(4):251–256

Codognet P, Diaz D (2001) Yet another local search method for constraint solving. In: International Symposium on Stochastic Algorithms: Foundations and Applications (SAGA), Springer, LNCS, vol 2264, pp 342–344

Craenen BG, Eiben A, van Hemert JI (2003) Comparing evolutionary algorithms on binary constraint satisfaction problems. IEEE Transactions on Evolutionary Computation 7(5):424–444

Crawford JM, Auton L (1993) Experimental results on the cross-over point in satisfiability problems. In: Proceedings of the National Conference on Artificial Intelligence, pp 22–28

Davenport A, Tsang E, Wang CJ, Zhu K (1994) Genet: A connectionist architecture for solving constraint satisfaction problems by iterative improvement. In: Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI), AAAI, vol 1, pp 325–330

Dorne R, Hao JK (1998) A new genetic local search algorithm for graph coloring. In: 5th International Conference on Parallel Problem Solving from Nature (PPSN), Springer, Lecture Notes in Computer Science, vol 1498, pp 745–754

Eiben A, van der Hauw J (1997) Solving 3-sat with adaptive genetic algorithms. In: Proceedings of the Fourth IEEE Conference on Evolutionary Computation, IEEE Press, pp 81–86

Eiben AE, Smith JE (2003) Introduction to Evolutionary Computing. Springer

Feo TA, Resende MG (1989) A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letter 8:67–71

Fleurent C, Ferland JA (1996) Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability. DIMACS Series in Discrete Mathematics and Theoretical Computer Science 26:619–652

Galinier P, Hao JK (1997) Tabu search for maximal constraint satisfaction problems. In: International Conference on Principles and Practice of Constraint Programming (CP), Springer, LNCS, vol 1330, pp 196–208

Galinier P, Hao JK (1999) Hybrid evolutionary algorithms for graph coloring. Journal of Combinatorial Optimization 3(4):379–397

Galinier P, Hao JK (2004) A general approach for constraint solving by local search. Journal of Mathematical Modelling and Algorithms 3(1):73–88

Galinier P, Hertz A, Zufferey N (2008) An adaptive memory algorithm for the k-coloring problem. Discrete Applied Mathematics 156(2):267–279

Gent IP, Walsh T (1993) Towards an understanding of hill-climbing procedures for sat. In: Proceedings of AAAI-93, pp 28–33

Glover F, Laguna M (1993) Tabu search. In: Reeves C (ed) Modern Heuristics Techniques for Combinatorial Problems, Blackwell Scientific Publishing, Oxford, UK, pp 70–141

Goëffon A, Richer J, Hao J (2008) Progressive tree neighborhood applied to the maximum parsimony problem. IEEE/ACM Transations on Computational Biology and Bioinformatics 5(1):136–145

Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning. Addison-Wesley

Gottlieb J, Voss N (1998) Improving the performance of evolutionary algorithms for the satisfiability problem by refining functions. In: Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature, Springer, Lecture Notes in Computer Science, vol 1498, pp 755–764

Gottlieb J, Marchiori E, Rossi C (2002) Evolutionary algorithms for the satisfiability problem. Evolutionary Computation 10:35–50

Hansen P, Mladenovic N (2001) Variable neighborhood search: Principles and applications. European Journal of Operational Research 130(3):449–467

Hao J (2012) Memetic algorithms in discrete optimization. In: Handbook of Memetic Algorithms, Studies in Computational Intelligence 379, pp 73–94

Hao JK, Dorne R (1994) An empirical comparison of two evolutionary methods for satisfiability problems. In: Proc. of IEEE Intl. Conf. on Evolutionary Computation, IEEE Press, pp 450–455

Hao JK, Dorne R (1996) Empirical studies of heuristic local search for constraint solving. In: International Conference on Principles and Practice of Constraint Programming (CP), Springer, LNCS, vol 1118, pp 194–208

van Hemert JI, Solnon C (2004) A study into ant colony optimization, evolutionary computation and constraint programming on binary constraint satisfaction problems. In: Evolutionary Computation in Combinatorial Optimization (EvoCOP 2004), Springer-Verlag, LNCS, vol 3004, pp 114–123

Hentenryck PV, Michel L (2005) Constraint-based local search. MIT Press

Hirsch EA (2005) Unitwalk: A new sat solver that uses local search guided by unit clause elimination. Annals of Mathematics and Artificial Intelligenc 24(1-4):91111

Holland JH (1975) Adaptation and artificial systems. University of Michigan Press

Hoos HH, Stützle T (2005) Stochastic local search, foundations and applications. Morgan Kaufmann

Hoos HH, Neumann F, Trautmann H (2017) Automated Algorithm Selection and Configuration (Dagstuhl Seminar 16412). Dagstuhl Reports 6(10):33–74

Hutter F, Tompkins DAD, Hoos HH (2002) Scaling and probabilistic smoothing: Efficient dynamic local search for sat. In: Proceedings of CP 2002, Principles and Practice of Constraints Programming, Springer Verlag, Lecture Notes in Computer Science, pp 233–248

Hutter F, Lindauer M, Balint A, Bayless S, Hoos HH, Leyton-Brown K (2017) The configurable SAT solver challenge (CSSC). Artificial Intellence 243:1–25

Ishtaiwi A, Thornton J, Sattar A, Pham DN (2005) Neighbourhood clause weight redistribution in local search for sat. In: Proceedings of CP 2005, pp 772–776

Jagota A, Sanchis LA (2001) Adaptive, restart, randomized greedy heuristics for maximum clique. Journal of Heuristics 7(6):565–585

Jong KD, Spears W (1989) Using genetic algorithms to solve np-complete problems. In: Intl. Conf. on Genetic Algorithms (ICGA'89), Fairfax, Virginia, pp 124–132

Jussien N, Lhomme O (2002) Local search with constraint propagation and conflict-based heuristics. Artificial Intelligence 139(1):21–45

Khichane M, Albert P, Solnon C (2008) Integration of ACO in a constraint programming language. In: 6th International Conference on Ant Colony Optimization and Swarm Intelligence (ANTS'08), Springer, LNCS, vol 5217, pp 84–95

Khichane M, Albert P, Solnon C (2010) Strong integration of ant colony optimization with constraint programming optimization. In: 7th International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming (CPAIOR), Springer, LNCS, vol 6140, pp 232–245

Lardeux F, Saubion F, Hao JK (2006) Gasat: a genetic local search algorithm for the satisfibility problem. Evolutionary Computation 14(2):223–253

Larranaga P, Lozano JA (2001) Estimation of Distribution Algorithms. A new tool for Evolutionary Computation. Kluwer Academic Publishers

Li CM, Huang WQ (2005) Diversification and determinism in local search for satisfiability. In: Proceedings of SAT 2005, Springer, Lecture Notes in Computer Science, vol 3569, pp 158–172

Lourenco HR, Martin O, Stützle T (2002) Handbook of Metaheuristics, Kluwer Academic Publishers, chap Iterated Local Search, pp 321–353

Lü Z, Hao JK (2010) A memetic algorithm for graph coloring. European Journal of Operational Research 200(1):235–244

Ma F, Hao J (2017) A multiple search operator heuristic for the max-k-cut problem. Annals of Operational Research 248(1-2):365–403

Malaguti E, Monaci M, Toth P (2008) A metaheuristic approach for the vertex coloring problem. INFORMS Journal on Computing 20(2):302–316

Mazure B, Sais L, Grégoire E (1997) Tabu search for sat. In: Proc. of the AAAI-97, pp 281–285

McAllester D, Selman B, Kautz H (1997) Evidence for invariants in local search. In: Proc. of the AAAI 97

Minton S, Johnston MD, Philips AB, Laird P (1992) Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. Artificial Intelligence 58:161–205

Moscato P (1999) Memetic algorithms: a short introduction. In: Corne D, Dorigo M, Glover F (eds) New Ideas in Optimization, McGraw-Hill Ltd., Maidenhead, UK., pp 219–234

Neri F, Cotta C, (Eds) PM (2012) Handbook of memetic algorithms, studies in computational intelligence 379. Springer

Neveu B, Trombettoni G, Glover F (2004) Id walk: A candidate list strategy with a simple diversification device. In: International Conference on Principles and Practice of Constraint Programming (CP), Springer Verlag, LNCS, vol 3258, pp 423–437

Nonobe K, Ibaraki T (1998) A tabu search approach to the constraint satisfaction problem as a general problem solver. European Journal of Operational Research 106:599–623

Pelikan M, Goldberg DE, Cantú-Paz E (1999) BOA: The Bayesian optimization algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99, Morgan Kaufmann Publishers, San Fransisco, CA, vol I, pp 525–532

Pham DN, Thornton J, Sattar A (2007) Building structure into local search for sat. In: Proceedings of IJCAI 2007

Porumbel DC, Hao JK, Kuntz P (2010) An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. Computers and Operations Research 37(10):1822–1832

Prestwich S (2005) Random walk with continuously smoothed variable weights. In: Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT 2005)

Resende MG, Ribeiro CC (2003) Handbook of Metaheuristics, Kluwer Academic Publishers, chap Greedy randomized adaptive search procedures, pp 219–249

Rossi C, Marchiori E, Kok (1999) A flipping genetic algorithm for hard 3-sat problems. In: Proc. of the Genetic and Evolutionary Computation Conference, vol 1, pp 393–400

Rossi C, Marchiori E, Kok (2000) An adaptive evolutionary algorithm for the satisfiability problem. In: Carroll Jea (ed) Proceedings of ACM Symposium on Applied Computing, ACM, New York, pp 463–469

Schuurmans D, Southey F, Holte R (2001) The exponential subgradient algorithm for heuristic boolean programming. In: Proceedings of AAAI 2001, pp 334–341

Selman B, Kautz H (1993) Domain-independent extensions to gsat: Solving large structured satisfiability problems. In: Proceedings of IJCAI-93, pp 290–295

Selman B, Kautz H (1994) Noise strategies for improving local search. In: Proc. of the AAAI 94, pp 337–343

Selman B, Levesque H, Mitchell D (1992) A new method for solving hard satisfiability problems. In: 10th National Conference on Artificial Intelligence (AAAI), pp 440–446

Selman B, Kautz HA, Cohen B (1994) Noise strategies for improving local search. In: Proceedings of the 12th National Conference on Artificial Intelligence, AAAI Press / The MIT Press, Menlo Park, CA, USA, pp 337–343

Shang Y, Wah BW (1998) A discrete lagragian based global search method for solving satisfiability problems. Journal of Global Optimization 12:61–100

Solnon C (2002) Ants can solve constraint satisfaction problems. IEEE Transactions on Evolutionary Computation 6(4):347–357

Solnon C (2008) Combining two pheromone structures for solving the car sequencing problem with Ant Colony Optimization. European Journal of Operational Research (EJOR) 191:1043–1055

Solnon C (2010) Constraint Programming with Ant Colony Optimization (232 pages). John Wiley and Sons

Sörensen K (2013) Metaheuristics — the metaphor exposed. International Transactions in Operational Research pp 1–16

Spears WM (1996) Simulated annealing for hard satisfiability problems. DIMACS Series in Discrete Mathematics and Theoretical Computer Science 26:533–558

Stützle T, Hoos HH (2000) $\mathscr{MAX}-\mathscr{MIN}$ Ant System. Journal of Future Generation Computer Systems, special isuue on Ant Algorithms 16:889–914

Thornton J, Pham DN, Bain S, Ferreira VJ (2004) Additive versus multiplicative clause weighting for sat. In: Proceeding of AAAI 2004, pp 191–196

Wallace RJ (1996) Analysis of heuristics methods for partial constraint satisfaction problems. In: International Conference on Principles and Practice of Constraint Programming (CP), Springer, LNCS, vol 1118, pp 308–322

Wu Q, Hao J, Glover F (2012) Multi-neighborhood tabu search for the maximum weight clique problem. Annals of Operational Research 196(1):611–634

Xu L, Hoos HH, Leyton-Brown K (2010) Hydra: Automatically configuring algorithms for portfolio-based selection. In: 24th AAAI Conference on Artificial Intelligence, pp 210–216

Young R, Reel A (1990) A hybrid genetic algorithm for a logic problem. In: Proc. of the 9th European Conf. on Artificial Intelligence, Stockholm, Sweden, pp 744–746

Zinflou A, Gagné C, Gravel M (2007) Crossover operators for the car sequencing problem. In: 7th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP), Lecture Notes in Computer Science, vol 4446, pp 229–239

Zlochin M, Birattari M, Meuleau N, Dorigo M (2004) Model-based search for combinatorial optimization: A critical survey. Annals of Operations Research 131:373–395