ISSN 1392 – 124X INFORMACINĖS TECHNOLOGIJOS IR VALDYMAS, 2004, Nr. 1(30)

AN OVERVIEW OF SOME HEURISTIC ALGORITHMS FOR COMBINATORIAL OPTIMIZATION PROBLEMS

Alfonsas Misevičius[1], Tomas Blažauskas[2], Jonas Blonskis[1], Jonas Smolinskas[1]

Kaunas University of Technology, Department of Practical Informatics[1], Department of Software Engineering[2], Studentų St. 50, LT–3031 Kaunas, Lithuania

Abstract. Heuristic algorithms (or simply heuristics) are methods that seek for high quality solutions within a reasonable (limited) amount of time without being able to guarantee optimality. They often come out as a result of imitation of the real world (physics, nature, biology, etc.). In this paper, we give an overview of some heuristic algorithms for combinatorial optimization problems. At the beginning, some definitions related to combinatorial optimization, as well as the principle (framework) and basic features of the heuristics for combinatorial problems are concerned. Then, several popular heuristic algorithms are discussed, namely: descent local search, simulated annealing, tabu search, genetic algorithms, ant algorithms, and iterated local search. The unified paradigms of these heuristics are given. Finally, we present some results of comparisons of these algorithms for the well-known combinatorial problem, the quadratic assignment problem.

Keywords: heuristic algorithms, descent local search, simulated annealing, tabu search, genetic algorithms, ant algorithms, iterated local search, combinatorial optimization, quadratic assignment problem.

Indroduction

The algorithms for combinatorial optimization (CO) problems may roughly be classified into so-called exact and non-exact approaches. The exact techniques (such as branch and bound, branch and cut, dynamic programming) aim to produce solutions that have proven optimality. Unfortunately, many combinatorial optimization problems belong to the class NP-hard and can not be solved to optimality within polynomially bounded computation time. So, heuristic algorithms (or simply heuristics) [47, 53, 56] have to be used in order to find near-optimal (locally optimal) solutions. Heuristic algorithms seek for high quality solutions at a reasonable computational time, but can not guarantee that a problem will be solved in terms of obtaining the exact solution. (It may not even be possible to state how close to optimality a particular heuristic solution is.) Heuristic algorithms can also be seen as intelligent techniques that are based upon human's intuition, which, in turn, often comes out as a result of analogies with the processes in the surrounding world (physics, nature, biology, etc.) [47].

The approaches we consider are not pure heuristics. They are rather general purpose meta-heuristics than "tailored" algorithms. The main categories of heuristic approaches, which we are going to overview, are as follows: descent local search (LS), simulated annealing (SA), tabu search (TS), genetic algorithms (GA), ant algorithms (AA), and iterated local search (ILS).

Before starting the next section, we introduce some very basic definitions related to the combinatorial (discrete) optimization. So, let $S=\{s_1, s_2, ..., s_i, ...\}$ be a set of solutions (a "solution space") of a combinatorial optimization problem. For the sake of clarity, let us consider the case where the solutions are based on permutations of the integers from 1 to n, i.e. $\forall s_i \in S$, $s_i=(s_i(1), s_i(2), ..., s_i(n)) \in \{1, 2, ..., n\}$. Here n is the problem size. (It is obvious that $|S|=n!$) Furthermore, let f: $S \to \mathrm{R}^1$ be an objective (cost) function; without loss of generality, we assume that f seeks a global minimum (i.e. the goal is to seek for such a solution $s^* \in S$ that $s^* = \arg\min_{s \in S} f(s)$). In addition, a neighbourhood function N: $S \to 2^S$ is given: it attaches for each $s \in S$ a set $N(s) \subseteq S$ – a set of neighbouring solutions of s. Each solution $s' \in N(s)$ can be reached from s by an operation called a move, and s is said to move to s' when such an operation is performed (usually, the move follows the objective function evaluation which is called a trial). As long as we operate upon permutation based solutions, we can define the neighbourhood function, N_λ, of order λ ($1<\lambda \le n$), in the following way: $N_\lambda(s) = \{s' \mid s' \in S,\ \rho(s,s') \le \lambda\}$, where s is a solution from S, and $\rho(s,s')$ is so-called Hamming distance between the current solution s and the neighbouring

one s': $\rho(s,s') = \sum_{i=1}^{n} \mathrm{sgn} \mid s(i) - s'(i) \mid$. If λ=2, one obtains 2-exchange neighbourhood function, which is often used in combinatorial problems.

The remaining part of this paper is organized as follows. In Section 1, the basic principles of heuristic algorithms are outlined. LS, SA, TS, GA, AA, and ILS approaches are discussed in Sections 2, 3, 4, 5, 6, 7, respectively. In Section 8, we present some experimental results. Finally, Section 9 completes the paper with concluding remarks.

1. Heuristic algorithms: the principle and basic features

Very briefly, the principle (framework) of the heuristic algorithms can be described as follows [56]. Start the search from an initial solution, maybe, randomly generated. Continue the search process by performing (in a sequential way) some transformations to the solutions, i.e. making moves from solutions to solutions taking into account the neighbourhood function. The moves are controlled – more precisely, the decisions about moves ("to move, or not to move") are taken – depending on the quality of solutions (the corresponding values of the objective function). If the decision is "positive", then the current solution is replaced by the neighbouring one, which will be used as a "starting point" for the subsequent trials (in addition, the best so far solution is saved in a memory); otherwise, the search is continued with the current solution. The whole process is continued until some termination criterion is satisfied. The best solution, saved in the memory (the best locally optimal solution), is regarded as the final (resulting) solution.

As we can see, the following basic features are pertinent for the heuristic algorithms:

- memory (pool of solutions);
- transformations of solutions (neighbourhood function);
- search (exploration) order;
- decision rule;
- termination criterion.

There exist many variants in the choice of these features. For example, one can maintain one solution (the corresponding algorithms are referred to as single-solution algorithms), or several solutions (a population of solutions) (these algorithms are referred to as population-based algorithms (see Section 5)). When designing transformation operators, much more variations and modifications are available. For instance, let us consider the permutation-based solutions with the neighbourhood function N_2. In this case, a move from the solution s to the solution s' can formally be defined by using a 2-way perturbation (transformation) operator p_{jk}: $S \to S$ ($j,k \in \{1, 2, ..., n\}$, $j \neq k$), which exchanges exactly two elements (i.e. ith and jth elements) in the current solution. (Notation $s' = s \oplus p_{jk}$ means that s' is obtained from s by applying the perturbation p_{jk}.). 2-way perturbations are quite good choice for some perturbation-based problems. However, more sophisticated perturbations are possible. They can be interpreted as "large" moves, i.e. moves in higher order neighbourhoods N_λ (λ>2). Randomly chosen "large" moves (known as mutations) are often used (see Section 7). Moreover, the transformations can be applied to more than one solution – not only a single solution, for example, in genetic algorithms, a special recombination operator is applied to the pair of solutions.

Two main alternatives exist when exploring the neighbours of the current solution. Firstly, choose the next potential solution in a pure random way. Secondly, explore the neighbourhood in a systematic manner, for example, in the case of the neighbourhood function N_2, one can maintain a sequence $\{p_{jk}\}$, where j changes from 1 to n–1, and k – from j+1 to n.

Regarding the decision rules, two basic directions are available; they may be regarded to as deterministic direction, and stochastic direction. In the first case, the decisions are unambiguously predefined by the current value of the objective function; they are ordinarily based on greedy strategies (for example, only moves to the better neighbouring solutions are performed (see Section 2). In the second case, the decisions are made according to some probability that depends on the difference of the objective function values (see Section 3).

Finally, the different termination criteria of heuristic algorithms can be applied. The common way is to use a fixed a priori number of trials, although the sophisticated intelligent stopping conditions are available.

2. Descent local search

Before considering the refined modern heuristics, we roughly characterize the classical algorithms, so-called descent local search heuristics (also known as hill climbing) [54]. These algorithms are relatively very simple, but effective enough in some cases; they may also be thought of as origin of the more intelligent optimization techniques. There are two types of descent LS: a) a "greedy descent" (GD), and b) a "steepest descent" (SD). In both cases, the decision – to replace the current solution by the new one (i.e. to move to the neighbouring solution), or not – is positive if only the new solution is definitely better than the current one (that is, the difference in the objective function values is negative ($\Delta f = f(s') - f(s) < 0$, where $s' \in N(s)$) – here, we assume that the goal is to minimize the objective function). The difference between these LS groups is as follows. In the "greedy descent", a move is performed to the

first better solution from the neighbourhood of the current solution. In the "steepest descent" the current solution is replaced by the best improving neighbouring solution. (The first strategy is entitled as a "first improvement", and the second one – as a "best improvement".) In both cases, the search process is continued until the current solution, s, is locally optimal, that is, no better solution exists in the neighbourhood ($\forall s' \in N(s)$: $f(s') \geq f(s)$).

Given the neighbourhood N_λ, the solution obtained by SD is regarded to as an optimal (λ-opt(imal)) solution with respect to this neighbourhood. However, the complexity of this algorithm is proportional to $O(n^\lambda)$. Therefore, most of descent LS algorithms (as well as other iterative search algorithms) usually operate on lower order neighbourhoods, like N_2 or N_3. The paradigm of descent local search is presented (in an algorithmic language like form) in Figure 1.

```
procedure descent_local_search
  // input: s° – initial solution; output: s – the locally optimal solution //
  s ← s°;
  while s not locally optimal do begin // main cycle //
    choose the solution s′ from the neighbourhood of s, N(s),
    in such a way that f(s′)–f(s)<0;
    s ← s′ // replace the current solution s by the new one s′ (make a move to a new solution) //
  end // while //
end // procedure //
```

Figure 1. Paradigm of descent local search

The solutions obtained by LS are of poor quality in most cases. In order to improve the quality, one can use multiple runs (single applications) of the descent LS algorithm. In other words, LS is repeated many times by applying it to some "starting points" (initial solutions), a different "point" each time. The best found locally optimal solution is declared as the resulting solution. The approach described is the well-known multi-start (multi-greedy) (MS) technique [42]. In the simplest case (also known as random multi-start (RMS)), the purely randomly generated solutions play a role of "starting points". However, heuristically constructed solutions may be used. More elaborated strategies are concerned in Section 7.

3. Simulated annealing

Simulated annealing originated in statistical mechanics. It is based on a Monte Carlo model that was used by Metropolis et al., 1953 [46], to simulate energy levels in cooling solids. Boltzmann's law was used to determine the probability of accepting a perturbation resulting in a change ΔE in the energy at the current temperature t, i.e. $\mathrm{P} = \begin{cases} 1, \Delta E < 0 \\ \mathrm{e}^{-\Delta E/Ct}, \Delta E \geq 0 \end{cases}$, where C is a Boltzmann's constant. Starting from 1982, many authors [7,12,36,37,38,55] applied simulated annealing to solve combinatorial optimization problems.

The principle of SA algorithm is simple [37]: start from a random solution. Given a solution s, select a neighbouring solution s' and compute the difference in the objective function values, $\Delta f = f(s') - f(s)$. If the objective function value is improved ($\Delta f < 0$), then replace the current solution by the new one. If $\Delta f \geq 0$, then accept a move with probability $\mathrm{P}(\Delta f) = \mathrm{e}^{-\Delta f/t}$, where t is the current temperature (Boltzmann's constant is not required when applying the algorithm to combinatorial problems). Regarding the above probabilistic acceptance, it is achieved by generating a random number in [0,1] and comparing it against the threshold $\mathrm{e}^{-\Delta f/t}$ (here, the exponential function plays a role of an acceptance function). The procedure is repeated until a stopping condition is satisfied, for example, a predefined number of trials have been performed. As a resulting solution, usually the "best so far" (BSF) solution (instead of so-called "where you are" (WYA) solution) is returned by the algorithm. The paradigm of SA is presented in Figure 2.

Note that SA algorithm can also be viewed as an algorithm that belongs to a broad class of so-called threshold algorithms (TAs) (for more details on TAs, see, for example, [22]).

SA algorithms differ each from other mainly with respect to a cooling (annealing) schedule, which, in turn, is specified by initial and final values of the temperature, as well as an updating function for changing the temperature. Perhaps, the most important thing is how the initial temperature t_0 is determined. If the initial value of the temperature is chosen too high, then too many bad uphill moves are accepted, while if it is too low, then the search will quickly drop into a local optimum without possibility to escape form it. Thus, an optimum initial temperature must be somewhere between these two extremes.

The temperature is not a constant, but changes over time according to the updating function. One of the popular updating functions (known as Lundy-Mees schedule [40]) is characterized by the following relation: $t_{k+1} = t_k / (1 + \beta t_k)$, where k=0, 1, ...; t_0=const; $\beta \ll t_0$. It is easy to relate the coefficient β and the number of trials, i.e. the schedule length, L, under condition that the initial and final values of the temperature (t_0, t_f) are predefined: $\beta = (t_0 - t_f) / L t_0 t_f$.

It should be noted that, in the state-of-the-art SA algorithms, the temperature rather oscillates than decreases monotonically, i.e. re-annealing – a sequence of heating and cooling – is considered instead of the straightforward annealing (see, for example, [7]).

In theory, the SA procedure should be continued until the final temperature t_f is zero, but in practice other stopping criteria are applied, for example: a) the value of the objective function has not decreased for a large number of consecutive trials; b) the number of accepted moves has become less than a certain small threshold for a large number of consecutive trials; c) a predefined number of trials has been executed.

For a more completed discussion on simulated annealing, the reader is referred to [1,2].

```
procedure simulated_annealing
  // input: s° – initial solution; output: s* – the best solution found //
  s ← s°; s* ← s°;
  determine the initial temperature t0; t := t0;
  repeat // main cycle //
    calculate Δf = f(s′)–f(s), where s′∈N(s);
    generate uniform random number r from the interval [0,1];
    if (Δf<0) or (r < e^(−Δf/t)) then s ← s′; // replace the current solution by the new one //
    if f(s)<f(s*) then s* ← s; // save the best so far solution //
    update the current temperature t
  until termination criterion is satisfied
end // procedure //
```

Figure 2. Paradigm of simulated annealing

4. Tabu search

Tabu search meta-heuristic was introduced by Hansen and Jaumard, 1987 [31], and Glover, 1989, 1990 [26,27]. Since that time, TS has been proven to be among the most powerful intelligent techniques for difficult optimization problems (see, for example, [4,30,33,57,60,62,63]). Briefly speaking, TS is based on the neighbourhood search with local-optima avoidance but in a rather deterministic way. The key idea of tabu search is to allow climbing moves when no improving move (neighbouring solution) exists. However, some moves have to be forbidden in order to avoid cycling. So, the tabu search starts from an initial solution s, maybe, randomly generated in S and moves repeatedly from a solution to a neighbouring one. At each step of the procedure, a set (subset) $\varphi(s)$ of the neighbouring solutions of the current solution s is considered and the move that improves most the objective function value f is chosen. If there are no improving moves, tabu search chooses one that least degrades the objective function, i.e. a move is performed to the best neighbour s' in $\varphi(s)$ (even if $f(s') > f(s)$).

In order to avoid returning to the local optimal solution just visited, the reverse move must be forbidden (prohibited). This is done by storing this move (or an attribute of the move) into a memory (or more precisely short-term-memory) managed like a circular list T and called a tabu list. The tabu list keeps information on the last h ($h = |T|$) moves which have been done during the search process (the parameter h is called a tabu list size). Thus, a move from s to s' is considered as tabu if it (or its attribute) is contained in the list T. This way of proceeding hinders the algorithm from returning to a solution reached in the last h steps. However, it might be worth returning after a while to a solution visited previously to search in another direction. Consequently, an aspiration criterion is introduced to permit the tabu status to be dropped under certain favourable circumstances. Typically, a tabu move from s to s' is permitted if $f(s') < f(s^*)$, where s^* is the best solution found so far. The resulting decision rule within TS may thus be described as follows: replace s by s', where $f(s') < f(s^*)$ or $s' = \arg\min_{s'' \in \varphi(s)} f(s'')$ and s' is not tabu.

The process is stopped as soon as a termination criterion is satisfied (for example, a fixed a priori number of cycles has been performed). The simplified scheme of TS is shown in Figure 3.

The tabu search algorithms differ mainly with respect to the basic ingredients discussed above, i.e. tabu list

and aspiration criterion. Other important features should be mentioned: a long-term-memory, a target analysis, intensification, as well as diversification mechanisms. The main forms of the tabu search are: deterministic tabu search (strict tabu search, fixed tabu search, reactive tabu search), stochastic tabu search (probabilistic tabu search, robust tabu search), tabu thresholding. See Glover and Laguna, 1997 [28]; Hertz et al., 1997 [32], for more details on TS technique.

```
procedure tabu_search
    // input: s° – initial solution; output: s* – the best solution found //
    s ← s°; s* ← s°;
    initialize the tabu list T;
    repeat // main cycle //
      given neighbourhood function N, tabu list T, and aspiration criterion,
      find the best possible solution s′∈N(s);
      s ← s′; // replace the current solution by the new one //
      insert the solution s (or its attribute, e.g. the move from s to s′)
      into the tabu list T;
      if f(s)<f(s*) then s* ← s; // save the best so far solution //
      update the tabu list T
    until termination criterion is satisfied
 end // procedure //
```

Figure 3. Paradigm of tabu search

5. Genetic algorithms

The original concepts of genetic algorithms, which are based on the biological process of natural selection, were developed as far back as 1975 by Holland [34]. The power of GAs has been demonstrated for many optimization problems, among them, continuous optimization [13], graph partitioning [10], job-shop problems [64], quadratic assignment problem [21,23,49], set covering problem [6], traveling salesman problem [45].

GA operates with some group (P) (called a population) of solutions (s_1, s_2, ..., s_i, ...) (called individuals) from S. (This is quite different from the above two approaches, which can be viewed as single-solution heuristics.) So, each individual (s_i) is associated with some fitness corresponding to the objective function value ($f(s_i)$). In case of minimization problem, the less the objective function value, the more fit the individual, and the larger is the probability that the individual will survive in evolution process. Over many generations, best fitting individuals tend to predominate, while less fit individuals tend to die off. Further, GAs can be characterized by the following features: a) a mechanism for selecting individuals (corresponding to solutions of the optimization problem) from the population; b) an operator for creating new individuals, i.e. offsprings by combining the information contained in the previous individuals; c) a procedure for generating new solution by random perturbations of the single previous solutions; d) a rule for updating the population (culling solutions from the current population). These features are referred to as selection, crossover (recombination), mutation, and culling (updating), respectively.

There exists a great variety in the choice of how to select, cross, mutate, and cull the individuals; moreover, additional features may be used, for example, one can incorporate other local improvement algorithm (simulated annealing, tabu search) into GA in order to improve solution obtained after crossover – these strategies are called hybrid genetic (or memetic) algorithms [51]. For details and various modifications of GAs, see, for example, [16,29,52]).

The generalized framework for the genetic algorithms is presented in Figure 4.

6. Ant algorithms

Ant algorithms were first proposed by Dorigo and co-authors [19] as some system of cooperating agents (a multi-agent system) for various optimization problems. Recent applications of AA cover problems like graph colouring [15], job-shop scheduling [14], quadratic assignment problem [25,61], traveling salesman problem [20], vehicle routing problem [24]. Ant algorithms were inspired by the observation of real ant colonies. While walking from the nest to food sources and vice versa, ants deposit on the ground a substance called pheromone, forming in this way a pheromone trail. The role of this trail is to guide ants toward the source of food (or to the nest). It has been shown that the quantity of pheromone left by an ant depends on the amount of food found, it is also obvious that the more the ants which reach the source of food, the stronger the pheromone left. Since the shorter paths

from the nest to food sources will be travelled at a higher rate than longer ones, the amount of pheromone will grow faster on the shorter ways (i.e. pheromone quantity will be in relation with the path length). Thus, when many paths are available (from the nest to food sources), a colony of ants may exploit the pheromone trails (left by the individual ants) to discover the shortest path, i.e. ants are able to optimize their path by means of the process described above.

A similar process can be transposed to combinatorial optimization: solutions of a problem can be built using a statistics (information) of solutions previously constructed. This statistics plays the role of the pheromone traces and it gives an higher weight to the better solutions. Such a model is able to build solutions of better quality than a procedure guided objective function evaluations only.

```
procedure genetic_algorithm
  // input: P⊆S – population of solutions (possibly, locally optimized); output: s* – the best solution found //
  s* ← arg min_{s∈P} f(s) ;  // save the best solution from the initial population //
  repeat // main cycle //
    for counter := 1 to #offsprings do begin
      select "parents" s′,s″ ∈ P ;
      s‴ ← crossover( s′,s″ );  // apply crossover to "parents", get an "offspring" //
      apply local improvement to s‴ ;  // this step is optional //
      P ← P ∪ { s‴ };  // insert the (optimized) "offspring" into the population //
      if f( s‴ )<f(s*) then s* ← s‴ ;  // save the best so far solution //
      apply mutations to some members of P
    end;  // for //
    update (cull) the population P
  until termination criterion is satisfied
end // procedure //
```

Figure 4. Paradigm of genetic algorithm

```
procedure ant_algorithm
  // input: none; output: s* – the best solution found //
  f* := ∞; initialize pheromone trails;
  repeat // main cycle //
    for i := 1 to #ants do begin
      construct a solution s_i (corresponding to ith ant) by using actual trails
          (and so-called "visibility");
      apply local improvement to s_i;  // this step is optional //
      if f(s_i)<f* then begin f* := f(s_i); s* ← s_i end  // save the best so far solution //
    end;  // for //
    update pheromone trails
  until termination criterion is satisfied
end // procedure //
```

Figure 5. Paradigm of ant algorithm

The analogies of real ants and combinatorial optimization are as follows: (a) the real ants' search area corresponds to the set of solutions of the combinatorial optimization problem; (b) the amount of food in a source corresponds to the objective function value; (c) the pheromone trails correspond to some (adaptive) memory.

Note that, during the construction of a new solution, the only information sources for an artificial ant are the objective function value and the history of a population of ants, i.e. the pheromone trails left by other ants (having already built solutions) during a certain number of trials. After having built a new solution, the artificial ant updates the adaptive memory (i.e. the trails) taking into consideration the

quality of the solution just built. (A 2-dimensional array (the matrix of traces), τ, may be an example of the adaptive memory; in this case, the entries τ_{ij} of τ measure the "desirability" of setting, for example, $s(i)=j$.) The above process is repeated in an iterative way. The process terminates after some large number of cycles has been performed.

The basic structure of an artificial ant system based algorithm – ant algorithm – is sketched in Figure 5.

For a more detailed study on ant algorithms, see, for example, [17, 18].

7. Iterated local search

Recently, a "reconstruct and improve" policy based approach known as an iterated local search has became very popular, although the concept of similar approach is more than 15 years old and goes back to Baum [5]. The approach called iterated Lin-Kernighan [35] has been very successfully applied to the traveling salesman problem, and still is one of the best heuristics for the TSP. Later, various modifications and enhancements of the basic idea have been proposed, among them, large step Markov chains [44], combined local search (chained local optimization) [43], variable neighbourhood search [50], iterated local search [39], "ruin and recreate (R&R)" principle based approach [59].

The key idea of ILS is to obtain better optimization results by a reconstruction (destruction) of an existing solution and a following improvement (rebuilding) procedure [39]. By applying this type of process in an iterative way one seeks for high quality solutions. So, in the first phase of the process, one reconstructs (mutates) the existing solution; the rationale is that continuing search from the reconstructed solution may allow to escape from a local optimum and to try to find better solutions. In the second phase, one tries to improve the solution just "ruined" as best as one can; hopefully, the new solution is better than the solution(s) obtained in the previous phase(s) of the improvement. (Usually, the improvement is a more sophisticated part of the method.)

There are a lot of different ways to reconstruct and, especially, to improve the solutions. So, we think of the ILS approach as a meta-heuristic – not a pure heuristic. For example, for the improvement, we can use any iterative (local) search concept based algorithm. In the simplest case, a hill-climbing (greedy descent or steepest descent) algorithm can be used; however, more refined approaches are possible, first of all, simulated annealing and tabu search algorithms discussed above.

The advantage of ILS over the well-known random multi-start (multi-greedy) approach (that is based on multiple starts of local improvements applied to randomly generated solutions) is that a better idea is to reconstruct (a part of) the current solution instead of "blind" generation of new solutions from scratch: indeed, the improvement of the (slightly) reconstructed (locally optimum) solution requires only a few steps to reach the next locally optimum solution, i.e., a new local optimum can be found very quickly – usually faster than when starting from a random solution.

The structure (paradigm) of the ILS metaheuristic is surprisingly simple. All one needs to do by creating the ILS principle based algorithm for a specific problem is to design three components (not taking into account an initial solution generation): 1) a solution improvement (local search) procedure, 2) a solution reconstruction (mutation) procedure, and 3) a candidate acceptance criterion (the last one is used to decide which solution is to be chosen for the reconstruction). The paradigm of the ILS is given in Figure 6.

```
procedure iterated_local_search
  // input: s° – initial solution; output: s* – the best solution found //
  s• ← local_search(s°); // improve the initial solution //
  s ← s•; s* ← s•;
  repeat // main cycle //
    s ← candidate_acceptance(s,s•); // choose the candidate for reconstruction (mutation) //
    s~ ← reconstruction(s); // reconstruct the current solution s, obtain the solution s~ //
    s• ← local_search(s~); // improve the reconstructed solution s~, obtain the improved solution s• //
    if f(s•)<f(s*) then s* ← s• // save the best so far solution //
  until termination criterion is satisfied
end // procedure //
```

Figure 6. Paradigm of iterated local search

8. Some computational experiments

In order to evaluate the efficiency of the above heuristic algorithms, an experiment has been carried out on the one of the hard combinatorial optimization problems, the quadratic assignment problem (QAP) [8,11,41]. The QAP is formulated as follows. Given two matrices $\boldsymbol{A} = (a_{ij})_{n\times n}$ and $\boldsymbol{B} = (b_{kl})_{n\times n}$ and the set Π of permutations of the integers from 1 to n, find a permutation $\pi = (\pi(1), \pi(2), ..., \pi(n)) \in \Pi$ that minimizes $z(\pi) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} b_{\pi(i)\pi(j)}$. It has been proved that the QAP is NP-hard [58]. Therefore, heuristic algorithms are to be used; they can find high quality solutions within reasonable computation times.

Table 1. Shorter run results of the comparison of the algorithms. The best results obtained are printed in bold face. CPU times per restart are given in seconds. 300 MHz PENTIUM computer was used in the experimentation

Instance	n	BKV	$\bar{\delta}$, $C_{1\%}/C_{bkv}$												CPU time
			MS		SA		TS		GA		AA		ILS		
chr25a	25	3796	21.81	0/0	35.09	0/0	12.43	1/1	**3.38**	2/2	18.80	0/0	7.85	1/1	*0.8*
els19	19	17212548	0.09	10/9	3.62	3/1	5.42	1/1	**0**		1.26	7/7	**0**		*0.4*
esc32a	32	130	7.69	0/0	2.00	2/2	10.15	0/0	**0.77**	6/6	4.46	0/0	0.92	5/5	*1.2*
kra30a	30	88900	2.22	1/1	1.23	2/2	1.66	1/1	**0.59**	7/5	1.60	2/2	0.66	6/5	*1.1*
nug30	30	6124	0.92	5/0	0.37	10/1	0.18	10/1	0.06	10/3	0.70	9/0	**0.05**	10/4	*1.1*
sko42	42	15812	1.09	4/0	0.26	10/0	0.09	10/4	**0.06**	10/4	0.81	8/0	**0.06**	10/2	*2.5*
ste36a	36	9526	2.83	0/0	8.01	0/0	1.82	3/0	**0.43**	10/2	1.82	3/0	0.44	10/1	*2.1*
tai60a	60	7208572	3.23	0/0	2.07	0/0	1.61	0/0	1.40	1/0	3.30	0/0	**1.11**	2/0	*8.0*
tai80b	80	818415043	2.11	1/0	1.00	6/0	5.48	0/0	**0.30**	9/0	1.01	4/0	0.59	8/0	*27.0*
wil100	100	273038	0.56	10/0	**0.06**	10/0	0.23	10/0	0.13	10/0	0.49	10/0	0.10	10/0	*60.0*

Table 2. Longer run results of the comparison of the algorithms. The best results obtained are printed in bold face. CPU times per restart are given in seconds. 300 MHz PENTIUM computer was used in the experimentation

Instance	n	BKV	$\bar{\delta}$, $C_{1\%}/C_{bkv}$												CPU time
			MS		SA		TS		GA		AA		ILS		
chr25a	25	379612.75	0/0		30.34	0/0	5.58	1/1	**0**		6.19	3/3	4.09	2/2	*7.0*
els19	19	17212548	**0**		1.04	7/1	**0**		**0**		**0**		**0**		*3.0*
esc32a	32	1304.00	0/0		0.15	9/9	0.92	4/4	**0**		1.38	5/5	**0**		*12.0*
kra30a	30	889000.98	3/3		**0**		0.40	7/7	**0**		0.80	5/5	0.13	9/9	*11.0*
nug30	30	61240.48	10/0		0.12	10/2	0.01	10/8	**0**		0.27	10/2	**0**		*11.0*
sko42	42	15812	0.71	10/0	0.16	10/1	0.01	10/7	**0**		0.18	10/1	**0**		*25.0*
ste36a	36	95261.86	0/0		7.68	0/0	0.07	10/5	**0.02**	10/8	0.23	10/1	**0.02**	10/8	*20.0*
tai60a	60	72085722.95	0/0		1.74	0/0	1.21	2/0	1.10	3/0	2.49	0/0	**0.71**	9/0	*70.0*
tai80b	80	8184150431.29	1/0		0.59	8/0	0.64	8/0	**0.01**	10/7	0.45	8/0	0.10	10/0	*270*
wil100	100	2730380.40	10/0		**0.02**	10/0	0.05	10/0	0.03	10/1	0.20	10/0	**0.02**	10/0	*600*

For the comparison, the following heuristic algorithms were used: 1) multi-start (MS) heuristic (adapted by the authors according to [3]); 2) simulated annealing (SA) algorithm [7]; 3) tabu search (TS) algorithm [60], 4) genetic algorithm (GA) [49]; 5) ant algorithm (AA) [61], and 6) iterated local search (ILS) algorithm [48]. (Note that, in GA and ILS algorithms, the tabu search based procedures are used as local improvement procedures. In fact, these algorithms should be considered as hybrid heuristics.) We tested the above algorithms on the well-known QAP instances from the QAP instances library QAPLIB [9]. The performance measures of the algorithms are: a) the average deviation from the best known solution – $\bar{\delta}$ ($\bar{\delta} = 100(\bar{z} - \breve{z})/\breve{z}$ [%], where $\bar{z}$ is the average objective function value over 10 restarts (i.e. single applications of the algorithm to a given instance), and

$\breve{z}$ is the best known value (BKV) of the objective function (BKVs are from [9])); b) the number of solutions that are within 1% optimality ($\overline{\delta} \le 1$) (over 10 restarts) – $C_{1\%}$; c) the number of the best known (pseudo-optimal) values (solutions) found – C_{bkv}.

Two variants of runs of algorithms have been performed: shorter runs and longer runs (in the first case, the run time was very limited; in the second case, much more time was allocated). The results of comparisons of the algorithms are presented in Tables 1 and 2. In addition, we give an illustration of typical "trajectories" of the objective function values for several of the algorithms examined (see Figure 7).

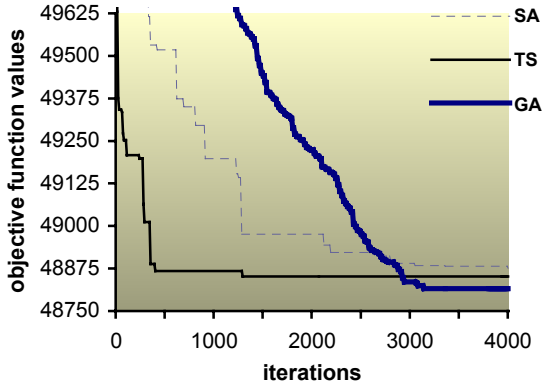

Figure 7. "Trajectories" of SA, TS and GA for the QAP

9. Concluding remarks

In this paper, a short overview of some heuristic algorithms for combinatorial optimization problems is given. The following heuristic methods have been concerned: descent local search, simulated annealing, tabu search, genetic algorithms, ant algorithms, iterated local search. In order to compare these algorithms, some experiments on the difficult combinatorial problem, the QAP, have been conducted. The results obtained from the experiments on various instances of the QAP show that, with respect to the performance measures used, the genetic algorithm and the iterated local search algorithm outperform all remaining heuristic algorithms, including the TS, SA and ant algorithms, on most of the QAP instances. In turn, the tabu search, simulated annealing and ant algorithms produce much more better results than the straightforward random multi-start algorithm. Clearly, for the sake of more accurate evaluation of the algorithms, more thorough and larger study should be done; in addition, the algorithms should be tested on the other combinatorial optimization problems. Nevertheless, some tendencies can be discovered. The attention should be paid to the genetic algorithms (especially, hybrid GAs) and iterated local search policy based algorithms. The idea of hybridization should further be exploited by investigating new procedures of both local improvement (intensification) and reconstruction (mutation) of solutions (diversification). The promising results could be achieved by applying effective hybrid genetic (memetic) algorithms, which incorporate fast efficient iterative (local) search, for example, the limited (iterated) tabu search.

References

[1] E.H.L. Aarts, J.H.M. Korst. Simulated Annealing and Boltzmann Machines. *Wiley, Chichester*, 1989.

[2] E.H.L. Aarts, J.H.M. Korst, P.J.M. van Laarhoven. Simulated annealing. *In E.H.L.Aarts, J.K.Lenstra (eds.), Local Search in Combinatorial Optimization, Wiley, Chichester,* 1997, 91–120.

[3] G.C. Armour, E.S. Buffa. A heuristic algorithm and simulation approach to relative location of facilities. *Management Science*, 1963, *Vol.*9, 294–304.

[4] R.Battiti, G.Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 1994, *Vol.*6, 126–140.

[5] E.B. Baum. Towards practical "neural" computation for combinatorial optimization problems. *In J.S.Denker (ed.) Neural networks for computing, American Institute of Physics, New York*, 1986, 53–58.

[6] J.E. Beasley, P.C. Chu. A genetic algorithm for the set covering problem. *European Journal of Operational Research*, 1996, *Vol.*94, 392–404.

[7] A. Bölte, U.W Thonemann. Optimizing simulated annealing schedules with genetic programming. *European Journal of Operational Research*, 1996, *Vol.*92, 402–416.

[8] R.E. Burkard, E. Çela, P.M. Pardalos, L. Pitsoulis. The quadratic assignment problem. *In D.Z.Du, P.M. Pardalos, (eds.), Handbook of Combinatorial Optimization*, *Vol.*3, *Kluwer, Dordrecht*, 1998, 241–337.

[9] R.E. Burkard, S. Karisch, F. Rendl. QAPLIB – a quadratic assignment problem library. *Journal of Global Optimization*, 1997, *Vol.*10, 391–403.

[10] T.N. Bui, B.R. Moon. Genetic algorithm and graph partitioning. *IEEE Transactions on Computers*, 1996, *Vol.*45, 841–855.

[11] E.Çela. The Quadratic Assignment Problem: Theory and Algorithms. *Kluwer, Dordrecht,* 1998.

[12] V. Cerný. A thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. *Tech. Report, Comenius University, Bratislava, CSSR*, 1982.

[13] R.Chelouah, P.Siarry. A continuous genetic algorithm designed for the global optimization of multimodal functions. *Journal of Heuristics*, 2000, *Vol.*6, 191–213.

[14] A.Colorni, M.Dorigo, V.Maniezzo, M.Trubian. Ant system for job-shop scheduling. *Belgian Journal of Operations Research, Statistics and Computer Science*, 1994, *Vol.*34, 39–53.

[15] D. Costa, A. Hertz. Ants can colour graphs. *Journal of the Operational Research Society*, 1997, *Vol.*48, 295–305.

[16] L. Davis. Handbook of Genetic Algorithms. *Van Nostrand, New York*, 1991.

[17] M. Dorigo, G. Di Caro. The ant colony optimization metaheuristic. *In D.Corne, M.Dorigo, F.Glover (eds.),*

New Ideas in Optimization, *McGraw-Hill, New York*, 1999, 11–32.

[18] M. Dorigo, G. Di Caro, L.M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 1999, *Vol*.5, 137–172.

[19] M. Dorigo, V. Maniezzo, A. Colorni. Positive feedback as a search strategy. *Tech. Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Italy*, 1991.

[20] M. Dorigo, V. Maniezzo, A. Colorni. The ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, 1996, *Vol*.26, 29–41.

[21] Z. Drezner. A new genetic algorithm for the quadratic assignment problem. *INFORMS Journal on Computing*, 2003, *in press*.

[22] G. Dueck, T. Scheuer. Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, 1990, *Vol*.90, 161–175.

[23] C. Fleurent, J.A. Ferland. Genetic hybrids for the quadratic assignment problem. *In P.M.Pardalos, H. Wolkowicz (eds.), Quadratic Assignment and Related Problems*. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, *Vol*.16*, AMS, Providence*, 1994, 173–188.

[24] L.M. Gambardella, E. Taillard, G. Agazzi. MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows. *In D.Corne, M.Dorigo, F.Glover (eds.), New Ideas in Optimization*, *McGraw-Hill, London*, 1999, 63–76.

[25] L.M. Gambardella, E. Taillard, M. Dorigo. Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society*, 1999, *Vol*.50, 167–176.

[26] F. Glover. Tabu search: part I. *ORSA Journal on Computing*, 1989, *Vol*.1, 190–206.

[27] F. Glover. Tabu search: part II. *ORSA Journal on Computing*, 1990, *Vol*.2, 4–32.

[28] F. Glover, M. Laguna. Tabu Search. *Kluwer, Dordrecht*, 1997.

[29] D.E. Goldberg. Genetic Algorithms in Search, Optimization and Machine Learning. *Addison-Wesley, Reading,* 1989.

[30] S. Hanafi, A. Freville. An efficient tabu search approach for the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, 1998, *Vol*.106, 93–100.

[31] P. Hansen, B. Jaumard. Algorithms for the maximum satisfiability problem. *RUTCOR Search Report* 43–87, *Rutgers University, USA*, 1987.

[32] A. Hertz, E. Taillard, D. de Werra. Tabu search. *In E.Aarts, J.K.Lenstra (eds.), Local Search in Combinatorial Optimization, Wiley, Chichester*, 1997, 121–136.

[33] A. Hertz, D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 1987, *Vol*.39, 345–351.

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

[35] D.S. Johnson. Local optimization and the traveling salesman problem. *In Proceedings of the 17th International Colloquium on Automata, Languages and Programming. Lecture Notes in Computer Science*, *Vol*.443*, Springer, Berlin,* 1990, 446–461.

[36] E.K. Karasakal, M. Köksalan. A simulated annealing approach to bicriteria scheduling problems on a single machine. *Journal of Heuristics*, 2000, *Vol*.6, 311–327.

[37] S. Kirkpatrick, C.D. Gelatt, Jr., M.P. Vecchi. Optimization by simulated annealing. *Science*, 1983, *Vol*.220, 671–680.

[38] M. Kolonko. Some new results on simulated annealing applied to the job shop scheduling problem. *European Journal of Operational Research*, 1999, *Vol.*113, 123–136.

[39] H.R. Lourenco, O. Martin, T. Stützle. Iterated local search. *In F.Glover, G.Kochenberger (eds.), Handbook of Metaheuristics, Kluwer, Norwell,* 2002, 321–353.

[40] M. Lundy, A. Mess. Convergence of an annealing algorithm. *Mathematical Programming*, 1986, *Vol*.34, 111–124.

[41] F. Malucelli. Quadratic assignment problems: solution methods and applications. *PhD Thesis, University of Pisa, Italy*, 1993.

[42] R. Marti. Multi-start methods. *In F.Glover, G.Kochenberger (eds.), Handbook of Metaheuristics, Kluwer, Norwell,* 2002, 355–368.

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

[44] O. Martin, S.W. Otto, E.W. Felten. Large-step Markov chains for the traveling salesman problem. *Complex Systems*, 1991, *Vol*.5, 299–326.

[45] P. Merz, B. Freisleben. Genetic local search for the TSP: new results. *Proceedings of the* 1997 *IEEE International Conference on Evolutionary Computation*, 1997, 159–164.

[46] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller. Equation of state calculation by fast computing machines. *Journal of Chemical Physics*, 1953, *Vol*.21, 1087–1092.

[47] Z. Michalewicz, D.B. Fogel. How to Solve It: Modern Heuristics. *Springer, Berlin-Heidelberg*, 2000.

[48] A. Misevicius. Ruin and recreate principle based approach for the quadratic assignment problem. *Lecture Notes in Computer Science: Genetic and Evolutionary Computation – GECCO-2003*, *Proceedings, Part I, E.Cantú-Paz et al. (eds.), Springer, Berlin-Heidelberg*, 2003, *Vol*.2723, 598–609.

[49] A. Misevicius. Genetic algorithm hybridized with ruin and recreate procedure: application to the quadratic assignment problem. *Knowledge-Based Systems*, 2003, *Vol*.16, 261–268.

[50] N. Mladenović, P. Hansen. Variable neighbourhood search. *Computers & Operations Research*, 1997, *Vol*.24, 1097–1100.

[51] P. Moscato. Memetic algorithms: a short introduction. *In D.Corne, M.Dorigo, F.Glover (eds.), New Ideas in Optimization, McGraw-Hill, London*, 1999, 219–234.

[52] **H. Mühlenbein.** Genetic algorithms. *In E.H.L.Aarts, J.K.Lenstra (eds.), Local Search in Combinatorial Optimization, Wiley, Chichester*, 1997, 137–171.

[53] **I.H. Osman, J.P. Kelly.** Meta-heuristics: an overview. *In I.H.Osman, J.P.Kelly (eds.), Meta-Heuristics: Theory and Applications, Kluwer, Norwell,* 1996, 1–21.

[54] **C.H. Papadimitriou, K. Steiglitz.** Combinatorial Optimization: Algorithms and Complexity. *Prentice-Hall, Englwood Cliffs,* 1982.

[55] **M. Randall, G. McMahon, S. Sugden.** A simulated annealing approach to communication network design. *Working Paper, School of Information Technology, Bond University, UK*, 1999.

[56] **C.R. Reeves.** Modern heuristic techniques. *In V.J. Rayward-Smith, I.H.Osman, C.R.Reeves, G.D.Smith (eds.). Modern Heuristic Search Methods, Wiley, Chichester,* 1996, 1–25.

[57] **E. Rolland, H. Pirkul, F. Glover.** Tabu search for graph partitioning. *Annals of Operations Reserch*, 1996, *Vol*.63, 209–232.

[58] **S. Sahni, T. Gonzalez.** P-complete approximation problems. *Journal of ACM*, 1976, *Vol*.23, 555–565.

[59] **G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt, G. Dueck.** Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, 2000, *Vol*.159, 139–171.

[60] **E. Taillard.** Robust taboo search for the QAP. *Parallel Computing*, 1991, *Vol*.17, 443–455.

[61] **E. Taillard.** FANT: fast ant system. *Tech. Report IDSIA-46-98, Lugano, Switzerland*, 1998.

[62] **P. Thomas, S. Salhi.** A tabu search heuristic for the resource constrained project scheduling problem. *Journal of Heuristics*, 1998, *Vol*.4, 123–139.

[63] **S. Voss.** Dynamic tabu search strategies for the traveling purchaser problem. *Annals of Operations Research*, 1996, *Vol*.63, 253–275.

[64] **T. Yamada, R. Nakano.** A genetic algorithm applicable to large-scale job-shop problems. *In R.Männer, B.Manderick (eds.), Parallel Problem Solving from Nature* 2, *North-Holland, Amsterdam*, 1992, 281–290.

DOI: 10.5755/j01.itc.30.1.11799