Consultant-Guided Search – A New Metaheuristic for Combinatorial Optimization Problems

Serban Iordache
SCOOP Software GmbH
Am Kielshof 29, 51105 Köln, Germany
siordache@acm.org

ABSTRACT
In this paper, we present Consultant-Guided Search (CGS), a new metaheuristic for combinatorial optimization problems, based on the direct exchange of information between individuals in a population. CGS is a swarm intelligence technique inspired by the way real people make decisions based on advice received from consultants. We exemplify the application of this metaheuristic to a specific class of problems by introducing the CGS-TSP algorithm, an instantiation of CGS for the Traveling Salesman Problem. To determine if our direct communication approach can compete with stigmergy-based methods, we compare the performance of CGS-TSP with that of Ant Colony Optimization algorithms. Our experimental results show that the solution quality obtained by CGS-TSP is comparable with or better than that obtained by Ant Colony System and MAX-MIN Ant System.

Categories and Subject Descriptors
I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search – *heuristic methods.*

General Terms
Algorithms.

Keywords
Metaheuristics, combinatorial optimization, swarm intelligence, traveling salesman problem.

1. INTRODUCTION
Many combinatorial optimization problems of both practical and theoretical importance are known to be NP-hard. Since exact algorithms are not feasible in such cases, heuristics are the main approach to tackle these problems. Metaheuristics are algorithmic templates used to specify problem-independent optimization strategies, which can be instantiated in order to define problem-specific heuristics. Some of the most successful metaheuristics conceived in the last two decades are swarm intelligence techniques [3] like Ant Colony Optimization (ACO) [6], Particle Swarm Optimization (PSO) [8] or Bee Colony Optimization (BCO) [12]. They are population-based methods that make use of the global behavior that emerges from the local interaction of individuals with one another and with their environment.

GECCO'10, July 7–11, 2010, Portland, Oregon, USA.

In this paper, we introduce Consultant-Guided Search (CGS), a new population-based metaheuristic for combinatorial optimization problems. CGS takes inspiration from the way people make decisions based on advice received from consultants. Human behavior is complex, but CGS uses virtual people that follow only simple rules. Furthermore, there is no centralized control structure in CGS, and the group behavior self-organizes. These properties allow us to regard CGS as a swarm intelligence metaheuristic.

Individuals in a swarm intelligence system can use direct communication or stigmergy to exchange information. Stigmergy is an indirect form of communication based on traces left in the environment by an individual. Currently, ACO algorithms are the most successful swarm intelligence techniques that use stigmergic communication. In contrast, CGS uses only direct communication between individuals, thus bearing some resemblance to BCO.

We illustrate the use of the CGS metaheuristic by applying it to the Traveling Salesman Problem (TSP). To this end, we propose a concrete CGS algorithm, called CGS-TSP, and report the results of its application to symmetric instances of the TSP. Most swarm intelligence techniques have been applied to TSP, but currently ACO algorithms outperform BCO [12] [13] and PSO [4] [10] for this class of problems. Therefore, we are interested if our direct communication approach can compete with stigmergy-based methods and we compare the performance of CGS-TSP with that of ACO algorithms. Our experimental results show that the solution quality obtained by CGS-TSP is comparable with or better than that obtained by the Ant Colony System (ACS) [5] and MAX-MIN Ant System (MMAS) [11].

This paper is organized as follows: in Section 2 we describe the CGS metaheuristic; in Section 3 we present the CGS-TSP algorithm and report our experimental results; in Section 4 we conclude the paper and present future research directions.

2. THE CGS METAHEURISTIC
In this section, after describing the CGS method, we present its formalization as a metaheuristic and analyze its operation.

2.1 Method Description
CGS is a population-based method. An individual of the CGS population is a virtual person, which can simultaneously act both as a client and as a consultant. As a client, a virtual person constructs at each iteration a solution to the problem. As a consultant, a virtual person provides advice to clients, in order to help them construct a solution.

The goal of a combinatorial optimization problem is to find values for discrete variables in order to obtain an optimal solution with respect to a given objective function. In CGS, a client constructs a

solution as a sequence of steps, choosing at each step a value for one of the discrete variables of the considered problem.

At the beginning of each iteration, a client chooses a consultant that will guide it during the construction of the current solution. The first factor considered when deciding which consultant to choose is its ***reputation***. The reputation of a consultant depends on the number of successes achieved by its clients. We say that a client achieves a ***success***, if it constructs a solution better than all solutions found up to that point by any client guided by the same consultant. In this case, the consultant's reputation will be incremented. Should a client construct a solution that is better than all solutions previously found by any client, irrespective of the consultant used, the consultant's reputation will receive a supplementary bonus. On the other hand, consultant's reputation fades over time. To keep its reputation, a consultant needs that its clients constantly achieve successes.

For each consultant, the algorithm keeps track of the best result obtained by any client working under its guidance. We use the term ***result*** in this context to refer to the value of the objective function applied to the considered solution. Based on these results, CGS maintains a ranking of the consultants. For a small number of consultants appearing at the top of this ranking, the algorithm prevents their reputations from sinking below a predefined level. This is motivated by the fact that the records (or at least the very good results) set by them in the past are still current, even if they happened long ago.

Besides reputation, another factor contributing to the choice of a consultant is client's ***personal preference***. The concretization of this concept is specific to each application of CGS to a given class of optimization problems.

Each consultant has a ***strategy***, which is used in order to guide its clients during the solution construction. If the consultant's reputation sinks below a minimum value, it will take a ***sabbatical leave***, during which it will stop offering advice to clients and it will instead start searching for a new strategy to use in the future. At each iteration, a virtual person on sabbatical leave constructs a new strategy, based on some heuristic. After a predefined period of time, the sabbatical ends and the virtual person selects the best strategy found during this period. This best strategy will be subsequently used to guide clients. At the end of the sabbatical, the consultant's reputation is reset to a predefined value.

At each step of an iteration, a client receives from the consultant chosen for this iteration a suggestion regarding the next action to be taken. Solution construction is a stochastic process in CGS, therefore the client will not always follow the consultant's recommendation. Usually, at each step there are several variants the client can choose from. The variant recommended by the consultant has a higher probability to be chosen, but the client may opt for one of the other variants, which it selects based on some heuristic.

When a client achieves a success, it means having obtained a result that is better than the result it would have obtained following the consultant's recommendations at each step. For this reason, the consultant adjusts its strategy in order to reflect the sequence of decisions taken by the client. This way, a consultant dynamically improves its strategy every time one of its clients achieves a success.

2.2 The Metaheuristic

The pseudocode that formalizes the CGS metaheuristic is shown in Figure 1.

A virtual person may be in one of the following modes: *normal* and *sabbatical*. During the initialization phase (lines 2-5), virtual people are created and placed in sabbatical mode. Based on its mode, a virtual person constructs at each iteration (lines 7-33) either a solution to the problem (line 13) or a consultant strategy (line 9). Optionally, a local optimization procedure (line 17) may be applied to improve this solution or consultant strategy.

The consultant used to guide the solution construction (line 11) is chosen based on its reputation and on the personal preference of the client. Since a virtual person act simultaneously as a client and as a consultant, it is possible for a client to choose itself as a consultant. The exact details of how reputation and personal preference are used in order to select a consultant are specific to each application of CGS to a particular class of problems. Similarly, the metaheuristic does not detail how a solution is constructed based on the strategy promoted by the consultant or how a strategy is constructed by a virtual person in sabbatical mode.

```
1  procedure CGSMetaheuristic()
2    create the set 𝒫 of virtual persons
3    foreach p ∈ 𝒫 do
4      setSabbaticalMode(p)
5    end foreach
6    while (termination condition not met) do
7      foreach p ∈ 𝒫 do
8        if actionMode[p] = sabbatical then
9          currStrategy[p] ← constructStrategy(p)
10       else
11         currCons[p] ← chooseConsultant(p)
12         if currCons[p] ≠ null then
13           currSol[p] ← constructSolution(p, currCons[p])
14         end if
15       end if
16     end foreach
17     applyLocalOptimization()                    // optional
18     foreach p ∈ 𝒫 do
19       if actionMode[p] = sabbatical then
20         if currStrategy[p] better than bestStrategy[p] then
21           bestStrategy[p] ← currStrategy[p]
22         end if
23       else
24         c ← currCons[p]
25         if c ≠ null and currSol[p] is better than all solutions
26           found by a client of c since last sabbatical then
27             successCount[c] ← successCount[c] + 1
28             strategy[c] ← adjustStrategy(c, currSol[p])
29         end if
30       end if
31     end foreach
32     updateReputations()
33     updateActionModes()
34   end while
35 end procedure
```

Figure 1. The CGS metaheuristic

After the construction phase, a virtual person in sabbatical mode checks if it has found a new best-so-far strategy (lines 20-22),

while a virtual person in normal mode checks if it has achieved a success and, if this is the case, it adjusts its strategy accordingly (lines 24-29). Again, the method used to adjust the strategy is specific to each instantiation of the metaheuristic.

At the end of each iteration, the reputation and action mode of each virtual person are updated (lines 32-33).

Figure 2 details how consultants' reputations are updated based on the successes achieved by their clients. A concrete application of the metaheuristic must specify how reputations fade over time (line 4) and how the reputations of top-ranked consultants are prevented from sinking below a predefined level (line 13).

```
1  procedure updateReputations()
2     foreach p ∈ 𝒫 do
3        if actionMode[p] = normal then
4           rep[p] ← applyReputationFading(rep[p])
5           rep[p] ← rep[p] + successCount[p]
6           if currSol[p] is better than best-so-far solution then
7              rep[p] ← rep[p] + bonus
8           end if
9           if rep[p] > maxReputation then
10             rep[p] ← maxReputation
11          end if
12          if isTopRanked(p) then
13             rep[p] ← enforceMinimumReputation(rep[p])
14          end if
15       end if
16    end foreach
17 end procedure
```

Figure 2. Procedure to update reputations

Figure 3 details how the action mode of each virtual person is updated: consultants whose reputations have sunk below the minimum level are placed in sabbatical mode, while consultants whose sabbatical leave has finished are placed in normal mode.

```
1  procedure updateActionModes()
2     foreach p ∈ 𝒫 do
3        if actionMode[p] = normal then
4           if rep[p] < minReputation then
5              setSabbaticalMode(p)
6           end if
7        else
8           sabbaticalCountdown ← sabbaticalCountdown – 1
9           if sabbaticalCountdown = 0 then
10             setNormalMode(p)
11          end if
12       end if
13 end procedure
```

Figure 3. Procedure to update action modes

Figure 4 and Figure 5 show the actions taken to place a virtual person in sabbatical or normal action mode.

```
1  procedure setSabbaticalMode(p)
2     actionMode[p] ← sabbatical
3     bestStrategy[p] ← null
4     sabbaticalCountdown ← sabbaticalDuration
5  end procedure
```

Figure 4. Procedure to set the sabbatical mode

```
1  procedure setNormalMode(p)
2     actionMode[p] ← normal
3     rep[p] ← initialReputation
4     strategy[p] ← bestStrategy[p]
5  end procedure
```

Figure 5. Procedure to set the normal mode

2.3 Method Analysis

The goal of a metaheuristic is to guide the solution search toward promising regions of the search space, where high-quality solutions are expected to be found. In CGS, the strategy of a consultant can be seen as a region of the search space that is advertised by this consultant. Because consultants with a higher reputation are more likely to be chosen by clients, it is important to ensure that the reputation of a consultant is in concordance with the probability to find high-quality solutions in the region of the search space that it advertises.

As the strategy of a consultant approaches a local or global optimum, the rate of achieving new successes decreases. This leads to a decrease in the consultant's reputation. As seen, CGS maintains a ranking of the consultants. This ranking is not based on reputation, but on the best result obtained until then by any client guided by the considered consultant. It is likely that the global optimum lies in one of the regions advertised by consultants appearing at the top of the ranking. Preventing the reputations of these consultants to sink below a specified level guarantees that the search will continue in the regions that most likely contain the global optimum.

A heuristic should keep a balance between the exploration of new regions in the search space and the exploitation of promising regions already found. In CGS, the sabbatical leave allows to abandon regions of the search space that are no longer promising and to start exploring new regions. During the sabbatical, at each iteration a new region of the search space is visited. At the end, the best region found is chosen to be advertised as the consultant's strategy. Because this region had not been exploited before, it is expected that its corresponding results are initially rather modest. If clients would choose consultants based on their results, a consultant that has just finished its sabbatical leave would have only a small chance to be chosen. Fortunately, consultants are chosen based on their reputation and the reputation of a consultant is reset at the end of the sabbatical. The reset value should be low enough to prevent the overexploitation of a region whose potential is still unknown, but high enough to allow a few clients to exploit this region. If the region selected during the sabbatical leave really has potential, a great number of successes will be achieved in the next period for this region, leading to a rapid increase in the consultant's reputation. This way, the consultant is likely to rise to the top of the ranking, thus ensuring that its reputation will not fade below a specified level and, consequently, that the region of the search space it advertises will be further exploited by clients.

Two aspects should be taken into account when choosing the rate at which the reputation fades over time. On the one hand, a too high rate leads to the premature termination of the exploitation of promising regions of the search space, thus preventing the convergence of the algorithm. On the other hand, a too low rate leads to stagnation, because it keeps reputations at high values for a long time, thus preventing consultants from taking a sabbatical leave in order to explore new regions of the search space.

3. APPLYING CGS TO THE TSP

In this section, we illustrate the use of the CGS metaheuristic by applying it to the TSP. To this end, we propose a concrete CGS algorithm, called CGS-TSP, and report the results of its application to symmetric instances of TSP, comparing the solution quality obtained by CGS-TSP to that obtained by Ant Colony System (ACS) [5] and MAX-MIN Ant System (MMAS) [11].

3.1 The CGS-TSP algorithm

The CGS-TSP algorithm introduced in this subsection exemplifies how the CGS metaheuristic can be used to solve a specific class of problems, in this case the TSP.

TSP [1] is the problem of a salesman, who is required to make a round-trip tour through a given set of cities, so that the total traveling distance is minimal. The problem can be represented by a graph $G = (V, E)$, where $V = \{v_1, \dots, v_n\}$ is the set of nodes and $E = \{(i,j): i,j \in V\}$ is the set of edges. Each edge $(i,j) \in E$ has an associated cost measure d_{ij}, which we will refer to as the distance between cities i and j. If $d_{ij} = d_{ji}$ for all $i,j \in V$, the problem is a symmetric TSP, otherwise it is an asymmetric TSP. CGS-TSP can be applied to symmetric instances of the TSP, but it can be easily adapted to also solve asymmetric instances.

In order to apply CGS to a particular class of problems, one must define the different concepts used by this metaheuristic (e.g., strategy, result, personal preference) in the context of the given class of problems. Then, one must decide how to implement the actions left unspecified by the CGS metaheuristic (e.g., *constructStrategy*, *constructSolution*, *chooseConsultant*).

Constructing a solution for the TSP means building a closed tour that contains each node of the graph only once. To avoid visiting a node several times, each virtual person in CGS-TSP keeps a list of the nodes already visited in the current iteration.

The ***strategy*** of a consultant is represented by a tour, which it advertises to its clients; the ***result*** of a tour is computed as the inverse of its length. Since both solution construction and strategy construction imply building a tour, the type of decision a virtual person has to make at each step is the same in both cases: it has to choose the next city to be visited. However, the rules used to make decisions in each of these two cases are different.

To restrict the number of choices available at each construction step, CGS-TSP uses candidate lists that contain for each city *i* the closest *cand* cities, where *cand* is a parameter. This way, the feasible neighborhood of a person *k* when being at city *i* represents the set of cities in the candidate list of city *i* that person *k* has not visited yet.

During the ***sabbatical leave***, consultants build strategies using a heuristic based only on the distances between the current city and the potential next cities. The rule used at each step to choose the next city is inspired by the pseudorandom proportional rule introduced by the ACS algorithm, but since CGS does not use pheromones (or other stigmergic communication), our rule involves only distances between cities. A virtual person *k* located at city *i* moves to a city *j* according to the following rule:

$$j = \begin{cases} argmin_{l \in \mathcal{N}_i^k}\{d_{il}\} & , if\ a \le a_0 \\ J & , otherwise \end{cases} \quad (1)$$

where:

- $\mathcal{N}_i^k$ is the feasible neighborhood of person *k* when being at city *i*.
- d_{il} is the distance between cities *i* and *l*.
- a is a random variable uniformly distributed in [0,1] and a_0 ($0 \le a_0 \le 1$) is a parameter.
- J is a random variable selected according to the probability distribution given by formula (2), where β is a parameter.

$$p_{ij}^k = \frac{(1/d_{ij})^\beta}{\sum_{l \in \mathcal{N}_i^k}(1/d_{il})^\beta} \quad (2)$$

In other words, with probability a_0 the person moves to the closest city in its feasible neighborhood, while with probability $(1 - a_0)$ it performs an exploration of the neighbor cities, biased by the distance to the city *i*.

At each step of the solution construction, a client receives from its consultant a recommendation regarding the next city to be visited. This recommendation is based on the tour advertised by the consultant. Let *i* be the city visited by the client *k* at a construction step of the current iteration. To decide what city to recommend for the next step, the consultant finds the position at which the city *i* appears in its advertised tour and identifies the city that precedes *i* and the city that succeeds *i* in this tour. If neither of these two cities are already visited by the client, the consultant recommends the one that is closest to city *i*. If only one of these two cities is unvisited, this one is chosen to be recommended. Finally, if both cities are already visited, the consultant is not able to make a recommendation for the next step.

The client does not always follow the consultant's recommendation. Again, a pseudorandom proportional rule is used to decide which city to visit at the next step:

$$j = \begin{cases} v & , if\ v \ne null\ \wedge q \le q_0 \\ argmin_{l \in \mathcal{N}_i^k}\{d_{il}\} & , if\ (v = null\ \vee q > q_0) \wedge b \le b_0 \\ J & , otherwise \end{cases} \quad (3)$$

where:

- v is the city recommended by the consultant for the next step.
- q is a random variable uniformly distributed in [0,1] and q_0 ($0 \le q_0 \le 1$) is a parameter.
- $\mathcal{N}_i^k$ is the feasible neighborhood of person *k* when being at city *i*.
- d_{il} is the distance between cities *i* and *l*.
- b is a random variable uniformly distributed in [0,1] and b_0 ($0 \le b_0 \le 1$) is a parameter.
- J is a random variable selected according to the probability distribution given by formula (2).

In other words, if a recommendation is available, the client moves with probability q_0 to the city recommended by its consultant; with probability $b_0(1 - q_0)$ it moves to the closest city in its feasible neighborhood; with probability $(1 - b_0)(1 - q_0)$ it performs an exploration of the neighbor cities, biased by the distance to the city *i*.

The two factors that influence the choice of a consultant are: consultant's ***reputation*** and client's ***personal preference***. In CGS-TSP the personal preference is given by the result of the consultant's advertised tour, that is, by the inverse of the advertised tour length. The probability to choose consultant *k* is given by formula (4):

$$p_k = \frac{reputation_k{}^{\alpha} result_k{}^{\gamma}}{\sum_{c \in \mathcal{C}} reputation_c{}^{\alpha} result_c{}^{\gamma}} \quad (4)$$

where:

- $\mathcal{C}$ is the set of all available consultants, that is, the set of all virtual people that are not in sabbatical mode.
- α is a parameter that determines the influence of the reputation.
- γ is a parameter that determines the influence of the result.

A client is allowed to choose itself as a consultant. Because the probabilities given by formula (4) do not depend on the client making the choice, the client index does not appear in this formula.

Every time a client achieves a success (i.e., it finds a tour shorter than the tour advertised by its consultant), the consultant updates its strategy, replacing its advertised tour with the tour constructed by the client.

At each iteration, the consultant's k reputation fades as given by formula (5):

$$reputation_k \leftarrow reputation_k(1 - r) \quad (5)$$

where r is the reputation fading rate. CGS-TSP adjusts its reputation fading rate according to the total number s of successes achieved during the last 1000 iterations by the best *fadingRanks* consultants, where *fadingRanks* is a parameter:

$$r = r_0 \left(1 + \frac{s}{\sqrt{1 + \left(\frac{s}{f}\right)^2}} \right) \quad (6)$$

The parameter r_0 gives the reputation fading rate for the case where no successes where achieved by the best *fadingRanks* consultants during the last 1000 iterations. The parameter f controls how the reputation fading rate increases with the number s of successes. In particular, we have:

$$\lim_{s \to \infty} r = r_0(1 + f) \quad (7)$$

Since it is difficult to estimate what values are appropriate for the parameter *f*, we compute its value based on the value of another parameter. Let us denote by *repdec(s)* the decrease in reputation after 1000 iterations in the hypothetical case that s remains constant during this period:

$$repdec(s) = (1 - r_{(s)})^{1000} \quad (8)$$

We introduce the parameter k_{1000} that indicates how much greater is the decrease in reputation for a very high number of successes than the decrease in the case when no successes were achieved:

$$repdec(0) = k_{1000} \cdot \lim_{s \to \infty} repdec(s) \quad (9)$$

From equations (6) to (9) we have:

$$f = \left(\frac{1}{r_0} - 1\right)\left(1 - \frac{1}{\sqrt[1000]{k_{1000}}}\right) \quad (10)$$

Using for example k_{1000}=10 and $r_0 = 10^{-5}$, we obtain $f \cong 230$.

In CGS-TSP the reputation value cannot exceed a maximum value specified by the parameter *maxReputation*, and the reputation of a top-ranked consultant cannot sink below the limit given by: initialReputation · bestSoFarTourLen/advertisedTourLen.

The value of the parameter *minReputation* referred in Figure 3 is fixed in this algorithm at 1, because only the differences between *minReputation* and the other reputation parameters are relevant for the algorithm.

The optional local optimization step referred in Figure 1 can be implemented in CGS-TSP by a local search procedure.

3.2 Implementation

As mentioned before, our goal is to find out if the direct communication approach used by CGS can compete with stigmergy-based methods like ACO. To allow a meaningful comparison between heuristics, we have created a software package containing Java implementations of CGS-TSP, ACS and MMAS algorithms. The software package is available as an open source project at http://swarmtsp.sourceforge.net/. At this address, we also provide all configuration files, problem instances and results files for the parameter tuning and for the experiments described in this paper.

We have tried to share as much code as possible between the implementations of CGS-TSP, ACS and MMAS, and we have taken care to keep the same level of code optimization while implementing the code portions specific to each algorithm. This way, we ensure that the relative differences in performance observed in our experiments reflect only the characteristics of the algorithms and are not due to implementation factors.

The implementation of the ACO algorithms is basically a port to Java of Thomas Stützle's ACOTSP program available at http://www.aco-metaheuristic.org/aco-code/.

3.3 Parameter tuning

For all algorithms, the experiments presented in the following subsection have been performed without local search and using candidate lists of length 20. In the first series of experiments, each algorithm run has been terminated after constructing 500000 tours. In the preliminary tuning phase, we have used these basic settings to identify parameter values that produce good results. In order to make a fair comparison between algorithms, we have performed parameter tuning not only for CGS-TSP, but also for ACS and MMAS.

3.3.1 CGS-TSP tuning

Because CGS-TSP is a completely new algorithm, we have performed the parameter tuning in several steps. First, we have searched for a good configuration in a set of candidate configurations that span a large range of values. After this coarse tuning step, we have performed a fine tuning, by using candidate configurations in a closer range around the configuration selected in the previous step. Finally, we have identified dependencies between two parameters or between a parameter and the size of the problem.

We have excluded from tuning the parameters that configure the sabbatical leave, because the purpose of the sabbatical is only to provide a decent strategy to start with and, in our opinion, the algorithm should not be very sensitive to these parameters. We

have chosen $a_0 = 0.9$ and, in order to keep the sabbatical leave relatively short, $sabbaticalDuration = 100$.

In the coarse and fine tuning steps we have used the paramILS configuration framework [7] to find good parameter settings. ParamILS executes an iterated local search in the parameter configuration space and it is appropriate for algorithms with many parameters, where a full factorial design becomes intractable. As training data for paramILS, we have generated a set of 500 Euclidean TSP instances with the number of cities uniformly distributed in the interval $[300, 500]$, and with coordinates uniformly distributed in a square of dimension 10000 x 10000.

In the next step, using as starting point the configuration found during the fine tuning, we have checked if there is a dependency between the optimum value for $protectedRanks$ and the number m of virtual persons used. For each value of m in the interval $[4, 20]$, we have tuned the value of $protectedRanks$ using the F-Race method and the previous training set. F-Race [1] is a racing configuration method that sequentially evaluates candidate configurations and discards poor ones as soon as statistically sufficient evidence is gathered against them. It is appropriate when the space of candidate configurations is relatively small, because at the start all candidate configurations are evaluated. Applying the linear least squares method to the optimum configurations found by F-Race, we have obtained the equation:

$$protectedRanks = 0.8 \cdot m - 2.5 \quad (11)$$

In the final step, using as starting point the configuration found during the fine tuning, with the value of $protectedRanks$ given by the equation (11), we have checked if there is a dependency between the optimum value for the number m of virtual persons and the number n of cities of a TSP instance. To this end, we have generated 9 sets of Euclidean TSP instances, with coordinates uniformly distributed in a square of dimension 10000 x 10000, each set comprising 100 instances. Each instance in the first set has a number n of 100 cities, in the second set 200 cities, and so on, until the last set with 900 cities. Using again F-Race and the linear least squares method, we have obtained the equation:

$$m = 3 + 1400/n \quad (12)$$

The parameter settings used in the experiments presented in the following subsection are shown in Table 1.

3.3.2 ACS tuning

ACS is a well-known algorithm, for which good parameter values are already available (see [6], p.71): m=10, ρ=0.1, ξ=0.1, q_0=0.9, $\beta \in [2, 5]$. We have performed a fine tuning of these parameters using paramILS and the same training set used in the coarse and fine tuning steps of CGS-TSP. The best configuration found, which has been used in the experiments presented in the following subsection, is: m=6, ρ=0.9, ξ=0.1, q_0=0.5, β=6.

3.3.3 MMAS tuning

As in the case of ACS, good parameter values are already available for MMAS (see [6], p.71): m=n, ρ=0.02, α=1, $\beta \in [2, 5]$. We have performed a fine tuning of these parameters using paramILS and the same training set used in the coarse and fine tuning steps of CGS-TSP. The best configuration found, which has been used in the experiments presented in the following subsection, is: m=n, ρ=0.1, α=1, β=5.

Some of the values found by paramILS during the tuning of ACS and MMAS differ significantly from the standard values given in [6]. For this reason, for ACS and MMAS, we have repeated the experiments described in the next subsection, using the standard settings. The results obtained using the tuned parameters outperform those obtained with the standard settings, thus confirming the effectiveness of paramILS.

Table 1. Parameter settings for CGS-TSP

Parameter	Value	Description
m	3+1400/n	number of virtual persons
b_0	0.3	solution construction parameter *see formula (3)*
q_0	0.98	solution construction parameter *see formula (3)*
α	1	reputation's relative influence *see formula (4)*
β	10	heuristic's relative influence *see formula (2)*
γ	4	result's relative influence *see formula (4)*
maxReputation	70	maximum reputation value
initialReputation	10	reputation value after sabbatical
bonus	15	reputation bonus for best-so-far tour
protectedRanks	0.8*m-2.5	protected top consultants
r_0	10^{-5}	basic reputation fading rate *see formula (6)*
fadingRanks	2	top consultants for fading rate *see formula (6)*
k_{1000}	20	reputation decrease factor *see formulae (9) and (10)*

3.4 Experimental results

To compare the performance of CGS-TSP with that of ACS and MMAS, we have applied these algorithms without local search to 15 symmetric instances from the TSPLIB benchmark library [9]. The number of cities of the TSP instances used in our experiments is between 150 and 1060. We have intentionally included TSP instances whose number of cities lies outside the range of values considered in the tuning phase (300 to 500), because a good algorithm is not problem dependent, and therefore it should not be very sensitive to the set of training instances used for tuning.

The experiments have been performed on an HP ProLiant with 8 x 2.33 GHz Intel(R) Xeon(R) CPUs and 16 GB RAM, running Red Hat Enterprise Linux 5.

In the first series of experiments performed to compare CGS-TSP with ACS and MMAS, we have stopped each run after constructing 500000 tours. Table 2 reports for each algorithm and TSP instance the best and mean percentage deviations from the optimal solutions over 25 trials, as well as the sample standard deviations of the means. The best results for each problem are in boldface. We also report for each problem the p-values of the one-sided Wilcoxon rank sum tests for the null hypothesis (H_0) that there is no difference between the solution quality of CGS-TSP and that of the competing algorithm, and for the alternative hypothesis (H_1) that CGS-TSP outperforms the considered algorithm. Applying the Bonferroni correction for multiple comparisons, we obtain the adjusted α–level: 0.05 / 15 = 0.0033. The p-values in boldface indicate the cases where the null hypothesis is rejected at this significance level.

Table 2. Comparison of CGS-TSP with ACS and MMAS. Runs are terminated after constructing 500000 tours.

Problem name	CGS-TSP			ACS				MMAS			
	Best (%)	Mean (%)	Mean stdev	Best (%)	Mean (%)	Mean stdev	p-value	Best (%)	Mean (%)	Mean stdev	p-value
kroA150	**0.098**	**0.659**	0.280	0.151	1.145	0.744	0.0067	0.222	0.759	0.253	0.2022
kroB150	**0.000**	0.510	0.304	**0.000**	0.881	0.512	0.0076	**0.000**	**0.140**	0.239	1.0000
si175	**0.000**	**0.074**	0.049	0.051	0.173	0.091	< **0.0001**	0.028	0.124	0.065	0.0045
kroA200	0.051	0.286	0.181	0.140	0.861	0.756	**0.0002**	**0.041**	**0.178**	0.113	0.9740
kroB200	**0.258**	**0.840**	0.345	0.265	1.141	0.740	0.1522	0.455	0.853	0.202	0.5019
pr299	**0.218**	**0.777**	0.491	0.826	1.805	0.574	< **0.0001**	0.494	0.790	0.267	0.0472
lin318	0.949	1.430	0.309	**0.305**	1.708	0.867	0.0462	0.404	**0.834**	0.302	1.0000
pr439	**0.632**	**1.518**	0.649	0.661	2.294	1.686	0.0880	1.054	2.216	0.432	< **0.0001**
pcb442	**0.825**	**1.610**	0.395	1.266	2.926	1.107	< **0.0001**	0.924	1.762	0.668	0.4257
att532	1.383	2.015	0.261	1.380	2.653	0.814	**0.0006**	**1.015**	**1.622**	0.338	1.0000
u574	**2.073**	**2.841**	0.396	2.412	5.651	2.025	< **0.0001**	2.078	3.205	0.541	0.0036
gr666	**1.705**	**2.933**	0.455	2.649	4.030	0.832	< **0.0001**	2.481	3.369	0.606	0.0045
u724	1.601	2.399	0.363	1.253	2.195	0.607	0.9467	**1.133**	**1.663**	0.326	1.0000
pr1002	3.406	**4.524**	0.608	**3.350**	5.263	1.721	0.0734	5.868	8.607	1.501	< **0.0001**
u1060	**3.461**	**4.352**	0.530	4.135	7.454	1.889	< **0.0001**	7.961	9.924	1.170	< **0.0001**

Using the one-sided Wilcoxon signed rank test, we compute the p-values for the null hypothesis (H_0) that there is no difference between the means of CGS-TSP and the means of the competing algorithm considered, and the alternative hypothesis (H_1) that the means of CGS-TSP are smaller than the means of the considered algorithm. The p-values obtained are: 0.00009 for ACS and 0.22287 for MMAS.

Therefore, the null hypothesis can be rejected for ACS at a high significance level, but cannot be rejected for MMAS. This means that CGS-TSP clearly outperforms ACS for runs terminated after constructing 500000 tours, but there is no statistically significant difference between CGS-TSP and MMAS for this kind of runs.

Using a given number of constructed tours as termination condition does not provide a fair comparison, because the algorithms have different complexities. A fairer approach is to stop the execution after a given CPU time interval. In our second series of experiments, we have limited to 60 seconds the CPU time allowed for each run. The results are presented in Table 3, which has the same structure as Table 2.

Again, using the one-sided Wilcoxon signed rank test, we compute the p-values for the null hypothesis (H_0) that there is no difference between the means of CGS-TSP and the means of the competing algorithm considered, and the alternative hypothesis (H_1) that the means of CGS-TSP are smaller than the means of the considered algorithm. The p-values obtained are: 0.00003 for ACS and 0.00269 for MMAS.

The null hypothesis can be rejected for both ACS and MMAS at a high significance level, which means that for runs terminated after 60 seconds CPU time, CGS-TSP clearly outperforms both ACS and MMAS.

Table 3. Comparison of CGS-TSP with ACS and MMAS. Runs are terminated after 60 seconds CPU time.

Problem name	CGS-TSP			ACS				MMAS			
	Best (%)	Mean (%)	Mean stdev	Best (%)	Mean (%)	Mean stdev	p-value	Best (%)	Mean (%)	Mean stdev	p-value
kroA150	**0.207**	**0.430**	0.177	0.256	1.344	0.676	< **0.0001**	0.211	0.558	0.228	0.0076
kroB150	**0.000**	0.287	0.319	**0.000**	0.641	0.549	0.0113	**0.000**	**0.010**	0.037	1.0000
si175	**0.000**	**0.036**	0.033	**0.000**	0.168	0.103	< **0.0001**	**0.000**	0.090	0.047	< **0.0001**
kroA200	**0.000**	**0.127**	0.099	**0.000**	0.585	0.549	**0.0001**	0.065	0.151	0.102	0.0434
kroB200	**0.000**	**0.573**	0.320	0.034	1.131	0.661	< **0.0001**	0.340	0.699	0.245	0.0460
pr299	**0.174**	**0.602**	0.350	0.583	1.647	0.698	< **0.0001**	0.291	0.643	0.283	0.1304
lin318	**0.385**	1.068	0.340	0.578	1.847	0.815	< **0.0001**	0.452	**0.940**	0.441	0.9339
pr439	**0.354**	**1.133**	0.531	0.850	2.382	1.542	**0.0002**	1.582	2.472	0.591	< **0.0001**
pcb442	**0.441**	**1.261**	0.363	1.266	2.953	0.923	< **0.0001**	1.321	2.967	1.096	< **0.0001**
att532	**1.358**	**1.862**	0.228	1.806	3.196	0.814	< **0.0001**	1.611	2.920	0.840	< **0.0001**
u574	**1.775**	**2.485**	0.336	2.834	6.004	2.033	< **0.0001**	3.325	5.659	1.094	< **0.0001**
gr666	**1.994**	**2.757**	0.358	2.778	6.149	2.294	< **0.0001**	5.848	7.480	0.971	< **0.0001**
u724	**1.090**	**2.232**	0.378	1.341	2.655	0.992	0.0174	7.860	9.584	0.952	< **0.0001**
pr1002	**3.010**	**4.277**	0.574	4.689	8.188	2.253	< **0.0001**	15.056	16.753	0.685	< **0.0001**
u1060	**3.603**	**4.644**	0.590	7.371	12.316	3.202	< **0.0001**	17.607	19.582	0.876	< **0.0001**

In our last series of experiments, we compare the development of the mean percentage deviation over 25 trials as a function of the CPU time for instances lin318 and u1060, over 10000 seconds.

In our previous experiments, CGS-TSP has achieved relatively poor results for lin318 and, as seen in Figure 6, MMAS is able to produce better solutions than CGS-TSP after less than 20 seconds. Nevertheless, at the end of our 10000 seconds interval, CGS-TSP succeeds to outperform MMAS, albeit by a very small margin.

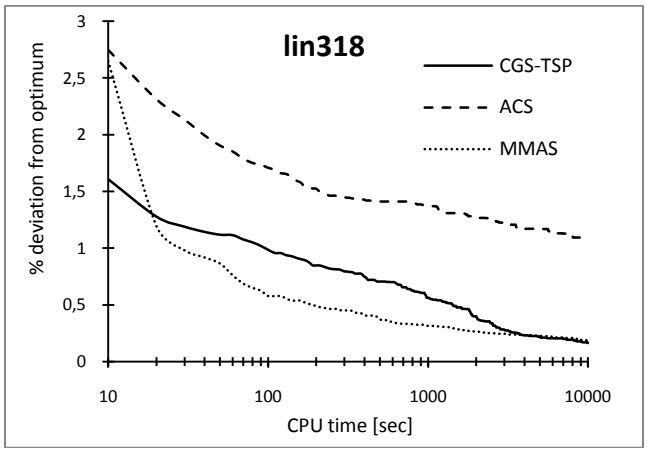

Figure 6. Mean percentage deviations for instance lin318.

In Figure 7, it can be observed that for u1060 CGS-TSP outperforms the other algorithms during the entire interval. MMAS is not able to reach the solution quality of CGS-TSP, although its performance improves significantly over time.

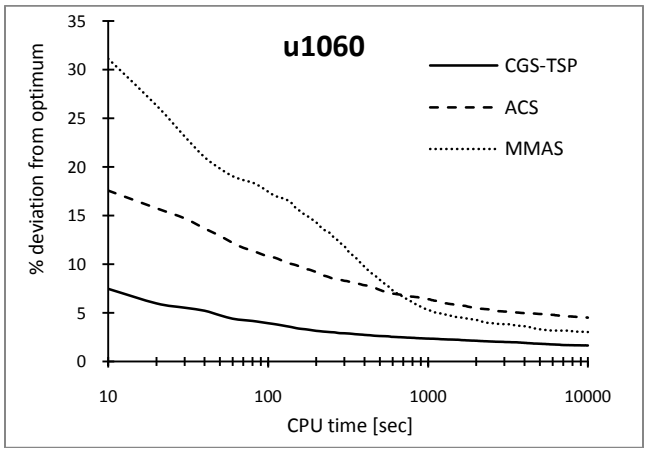

Figure 7. Mean percentage deviations for instance u1060.

4. CONCLUSIONS AND FUTURE WORK

In this paper, we have introduced CGS, a novel metaheuristic for combinatorial optimization problems, based on the direct exchange of information between individuals in a population, and we have devised the CGS-TSP algorithm, an instantiation of CGS for the TSP. Experimental results show that our direct communication approach can compete with the best ACO algorithms. Although these results are very encouraging, we regard CGS-TSP rather as a proof of concept application of CGS, meant to demonstrate the potential of this metaheuristic.

Nevertheless, we plan to improve CGS-TSP, making it able to solve asymmetric TSP instances, and also to study the performance of CGS-TSP combined with local search. Further, we will study the influence of the different components of CGS-TSP on its performance.

An important goal of our future research is to apply CGS to other combinatorial optimization problems in order to identify classes of problems for which it could achieve state of the art performance. We will also study the possibility to adapt the metaheuristic for continuous optimization problems.

Furthermore, we plan to analyze the viability of a hybrid approach that combines CGS and ACO, thus benefiting of both direct communication and stigmergy.

5. REFERENCES

[1] Applegate, D., Bixby, R., Chvatal, V. and Cook, W. *The traveling salesman problem: A computational study*. Princeton, NJ: Princeton University Press, 2007.

[2] Birattari, M., Stützle, T., Paquete, L. and Varrentrapp, K. *A racing algorithm for configuring metaheuristics*. Proceedings of GECCO 2002, 11-18.

[3] Bonabeau, E., Dorigo, M. and Theraulaz, G. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, USA, 1999.

[4] Clerc, M. Discrete Particle Swarm Optimization, illustrated by the Traveling Salesman Problem. In Onwubolu, G.C., Babu, B.V. (eds.) *New Optimization Techniques in Engineering*, Springer, 2004, 219-239.

[5] Dorigo, M. and Gambardella, L. M. *Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem*. IEEE Transactions on Evolutionary Computation, vol. 1, no. 1, 1997, 53-66.

[6] Dorigo, M. and Stützle, T. *Ant Colony Optimization*. The MIT Press, 2004.

[7] Hutter, F., Hoos, H.H., Leyton-Brown, K. and Stützle, T. *ParamILS: An Automatic Algorithm Configuration Framework*. Journal of Artificial Intelligence Research (JAIR), vol. 36, October 2009, 267-306.

[8] Kennedy, J. and Eberhart, R. *Swarm Intelligence*. Morgan Kaufmann, San Francisco, CA, 2001.

[9] Reinelt, G. *TSPLIB - A Traveling Salesman Problem Library*, ORSA Journal on Computing, vol. 3, no. 4, 1991, 376-384.

[10] Shi, X., Liang, Y., Lee, H., Lu, C., and Wang, Q. Particle swarm optimization-based algorithms for TSP and generalized TSP. In *Information Processing Letters*, vol. 103, no. 5, August 2007, 169-176.

[11] Stützle, T. and Hoos, H. H. *MAX-MIN Ant System*. Future Generation Computer Systems, 16(8), 2000, 889-914.

[12] Teodorovic, D. and Dell'Orco, M. Bee colony optimization - A cooperative learning approach to complex transportation problems. In *Advanced OR and AI Methods in Transportation*, 2005, 51—60.

[13] Wong, L., Low, M. and Chong, C. A Bee Colony Optimization Algorithm for Traveling Salesman Problem. In *Proceedings of Second Asia International Conference on Modelling & Simulation (AMS 2008)*, 2008, 818-823.