Ant-CSP:
an Ant Colony Optimization Algorithm for the Closest String Problem

Simone Faro
and
Elisa Pappalardo
(speaker)

January, 26 2010

Summary

Summary

1. Closest String Problem
2. Heuristic Algorithms
3. Ant Colony Optimization
4. Ant-CSP
5. Experimental Results

Summary

Summary

Summary

Closest String Problem

Problem Definition

Given

- a finite alphabet, Σ;
- a finite set of n strings, $S=\{s_1, s_2, ..., s_n\}$, each of length m,

the CLOSEST STRING PROBLEM for S is to find a string t over Σ, of length m, that minimizes the Hamming distance

$$H(t, S) = max_{s \in S} H(t, s).$$

Problem Definition

Given

- a finite alphabet, Σ;
- a finite set of n strings, $S=\{s_1, s_2, ..., s_n\}$, each of length m,

the CLOSEST STRING PROBLEM for S is to find a string t over Σ, of length m, that minimizes the Hamming distance

$$H(t, S) = max_{s \in S} H(t, s).$$

Problem Definition

Given

- a finite alphabet, Σ;
- a finite set of n strings, $S=\{s_1, s_2, ..., s_n\}$, each of length m,

the CLOSEST STRING PROBLEM for S is to find a string t over Σ, of length m, that minimizes the Hamming distance

$$H(t, S) = max_{s \in S} H(t, s).$$

Problem Definition

Given

- a finite alphabet, Σ;
- a finite set of n strings, $S=\{s_1, s_2, ..., s_n\}$, each of length m,

the CLOSEST STRING PROBLEM for S is to find a string t over Σ, of length m, that minimizes the Hamming distance

$$H(t, S) = max_{s \in S} H(t, s).$$

Applications

Recently, the CSP problem has received much attention, especially in computational biology and coding theory.

- In molecular biology, such problem finds applications, for instance, in genetic drug target and genetic probes design [Lanctot *et al.*, 1999], in locating binding sites [Stormo & Hartzell, 1989, Hertz *et al.*, 1990];
- in coding theory, to determine the best way to encode a set of messages [Gasieniec *et al.*, 1999, Frances & Litman, 1997, Roman, 1992].

Applications

Recently, the CSP problem has received much attention, especially in computational biology and coding theory.

- In molecular biology, such problem finds applications, for instance, in genetic drug target and genetic probes design [Lanctot *et al.*, 1999], in locating binding sites [Stormo & Hartzell, 1989, Hertz *et al.*, 1990];
- in coding theory, to determine the best way to encode a set of messages [Gasieniec *et al.*, 1999, Frances & Litman, 1997, Roman, 1992].

Applications

Recently, the CSP problem has received much attention, especially in computational biology and coding theory.

- In molecular biology, such problem finds applications, for instance, in genetic drug target and genetic probes design [Lanctot *et al.*, 1999], in locating binding sites
  [Stormo & Hartzell, 1989, Hertz *et al.*, 1990];
- in coding theory, to determine the best way to encode a set of messages
  [Gasieniec *et al.*, 1999, Frances & Litman, 1997, Roman, 1992].

The CSP problem is NP-hard

[Frances & Litman, 1997] have proved the NP-hardness of the problem for binary codes.

A successful strategy for approaching these problems is given by heuristic algorithms.

The CSP problem is NP-hard

[Frances & Litman, 1997] have proved the NP-hardness of the problem for binary codes.
A successful strategy for approaching these problems is given by heuristic algorithms.

Heuristic algorithms

Heuristic algorithms

Heuristic algorithms do not guarantee an optimal solution, but in general, they are able to provide a good feasible solution, i.e. a solution with a "value close" to the optimum.

Metaheuristic Algorithms and Nature

Metaheuristics represent a subclass of heuristic algorithms.
They are an extension of local search algorithms, where appropriate techniques are introduced aimed at preventing the termination of the algorithm in a local optimum.
Some metaheuristic algorithms are inspired by nature.

Metaheuristic Algorithms and Nature

Metaheuristics represent a subclass of heuristic algorithms.
They are an extension of local search algorithms, where appropriate techniques are introduced aimed at preventing the termination of the algorithm in a local optimum.
Some metaheuristic algorithms are inspired by nature.

Metaheuristic Algorithms and Nature

Metaheuristics represent a subclass of heuristic algorithms.
They are an extension of local search algorithms, where appropriate techniques are introduced aimed at preventing the termination of the algorithm in a local optimum.
Some metaheuristic algorithms are inspired by nature.

Ant Colony Optimization

- The new proposed approach for the CLOSEST STRING PROBLEM is based on ANT COLONY OPTIMIZATION (ACO) metaheuristic [Dorigo, 1992, Dorigo *et al.*, 1999].
- ACO is a multi-agent approach to difficult combinatorial optimization problems, like the Traveling Salesman Problem (TSP) and the Quadratic Assignment Problem (QAP).

- The new proposed approach for the Closest String Problem is based on Ant Colony Optimization (ACO) metaheuristic [Dorigo, 1992, Dorigo *et al.*, 1999].
- ACO is a multi-agent approach to difficult combinatorial optimization problems, like the Traveling Salesman Problem (TSP) and the Quadratic Assignment Problem (QAP).

Ants behaviour

ACO algorithms were inspired by the observation of real ant colonies, in particular, by the observation of their foraging behaviour:

- once a food source has been found, ants always seek the shortest and easiest path to return to their nest;
- while walking from nest to the food sources, and vice versa, ants deposit on the ground a substance called pheromone, forming in this way a pheromone trail;
- ants can smell pheromone *(stigmergy)* and, when choosing their way, they tend to choose, in probability, paths marked by strong pheromone concentrations.

Ants behaviour

ACO algorithms were inspired by the observation of real ant colonies, in particular, by the observation of their foraging behaviour:

- once a food source has been found, ants always seek the shortest and easiest path to return to their nest;
- while walking from nest to the food sources, and vice versa, ants deposit on the ground a substance called pheromone, forming in this way a pheromone trail;
- ants can smell pheromone *(stigmergy)* and, when choosing their way, they tend to choose, in probability, paths marked by strong pheromone concentrations.

Ants behaviour

ACO algorithms were inspired by the observation of real ant colonies, in particular, by the observation of their foraging behaviour:

- once a food source has been found, ants always seek the shortest and easiest path to return to their nest;
- while walking from nest to the food sources, and vice versa, ants deposit on the ground a substance called pheromone, forming in this way a pheromone trail;
- ants can smell pheromone *(stigmergy)* and, when choosing their way, they tend to choose, in probability, paths marked by strong pheromone concentrations.

Ants behaviour

ACO algorithms were inspired by the observation of real ant colonies, in particular, by the observation of their foraging behaviour:

- once a food source has been found, ants always seek the shortest and easiest path to return to their nest;
- while walking from nest to the food sources, and vice versa, ants deposit on the ground a substance called pheromone, forming in this way a pheromone trail;
- ants can smell pheromone *(stigmergy)* and, when choosing their way, they tend to choose, in probability, paths marked by strong pheromone concentrations.

It has been experimentally proved that pheromone trail behavior can give rise to the emergence of shortest paths, because on these paths pheromone density is higher [Deneubourg *et al.*, 1990].

Figure: Binary bridge experiment

From Nature to Optimization

The ANT COLONY OPTIMIZATION brings the pheromone and social behavior concepts from nature to discrete optimization problems.

Similarities with real ants

- Colony of cooperating individuals.
- Pheromone trails and stigmergy.
- Shortest path searching.
- Stochastic and myopic state transition policy.

Similarities with real ants

- Colony of cooperating individuals.
- Pheromone trails and stigmergy.
- Shortest path searching.
- Stochastic and myopic state transition policy.

Similarities with real ants

- Colony of cooperating individuals.
- Pheromone trails and stigmergy.
- Shortest path searching.
- Stochastic and myopic state transition policy.

Similarities with real ants

- Colony of cooperating individuals.
- Pheromone trails and stigmergy.
- Shortest path searching.
- Stochastic and myopic state transition policy.

Differences with real ants

- Artificial ants live in a discrete world.
- Artificial ants have an internal state.
- The amount of pheromone in ACO algorithms is proportional to the quality of the solution.
- Artificial ants timing in pheromone laying is problem dependent.

Differences with real ants

- Artificial ants live in a discrete world.
- Artificial ants have an internal state.
- The amount of pheromone in ACO algorithms is proportional to the quality of the solution.
- Artificial ants timing in pheromone laying is problem dependent.

Differences with real ants

- Artificial ants live in a discrete world.
- Artificial ants have an internal state.
- The amount of pheromone in ACO algorithms is proportional to the quality of the solution.
- Artificial ants timing in pheromone laying is problem dependent.

Differences with real ants

- Artificial ants live in a discrete world.
- Artificial ants have an internal state.
- The amount of pheromone in ACO algorithms is proportional to the quality of the solution.
- Artificial ants timing in pheromone laying is problem dependent.

Ant-CSP

Ant-CSP

The ACO metaheuristic has two main application fields:

- NP-hard problems,
- and shortest path problems.

As the CSP problem is NP-hard, and searching a closest string can be viewed as finding a minimum path, it is natural to apply the ACO heuristic to the CSP problem. This is what we did.

Ant-CSP

The ACO metaheuristic has two main application fields:

- NP-hard problems,
- and shortest path problems.

As the CSP problem is NP-hard, and searching a closest string can be viewed as finding a minimum path, it is natural to apply the ACO heuristic to the CSP problem. This is what we did.

Ant-CSP

The ACO metaheuristic has two main application fields:

- NP-hard problems,
- and shortest path problems.

As the CSP problem is NP-hard, and searching a closest string can be viewed as finding a minimum path, it is natural to apply the ACO heuristic to the CSP problem. This is what we did.

Ant-CSP

The ACO metaheuristic has two main application fields:

- NP-hard problems,
- and shortest path problems.

As the CSP problem is NP-hard, and searching a closest string can be viewed as finding a minimum path, it is natural to apply the ACO heuristic to the CSP problem. This is what we did.

Ant-CSP Algorithm 1/3

1. At each iteration, u artificial ants are generated;
2. each of them builds its closest string by moving on a $|\Sigma| \times m$ matrix, one character at time;
   - each location of the matrix, T_{ij}, $1 \leq i \leq |\Sigma|$ and $0 \leq j \leq m-1$, mantains the pheromone trail for the i-th character at the j-th position of the string.

Ant-CSP Algorithm 1/3

1. At each iteration, u artificial ants are generated;
2. each of them builds its closest string by moving on a $|\Sigma| \times m$ matrix, one character at time;
   - each location of the matrix, T_{ij}, $1 \leq i \leq |\Sigma|$ and $0 \leq j \leq m-1$, mantains the pheromone trail for the i-th character at the j-th position of the string.

Ant-CSP Algorithm 1/3

1. At each iteration, u artificial ants are generated;
2. each of them builds its closest string by moving on a $|\Sigma| \times m$ matrix, one character at time;
   - each location of the matrix, T_{ij}, $1 \leq i \leq |\Sigma|$ and $0 \leq j \leq m-1$, mantains the pheromone trail for the i-th character at the j-th position of the string.

Ant-CSP Algorithm 2/3

3. The evaluation function is the *maximum Hamming distance* between the current solution and the set of input strings.
4. Once all the ants have built a solution, pheromone evaporation is performed:
   - each of the matrix location T_{ij}, $1 \leq i \leq |\Sigma|, 0 \leq j \leq m-1$, is decremented by a constant factor.

Ant-CSP Algorithm 2/3

3. The evaluation function is the *maximum Hamming distance* between the current solution and the set of input strings.
4. Once all the ants have built a solution, pheromone evaporation is performed:
   - each of the matrix location T_{ij}, $1 \leq i \leq |\Sigma|, 0 \leq j \leq m-1$, is decremented by a constant factor.

Ant-CSP Algorithm 2/3

3. The evaluation function is the *maximum Hamming distance* between the current solution and the set of input strings.
4. Once all the ants have built a solution, pheromone evaporation is performed:
   - each of the matrix location T_{ij}, $1 \leq i \leq |\Sigma|, 0 \leq j \leq m-1$, is decremented by a constant factor.

ANT-CSP Algorithm 3/3

5. An *elitist strategy* is used to update pheromone trails:
   - pheromone trails increment is proportional to the distance of the current string from the input set, according to the rule:

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \left(1 - \frac{HD}{m}\right).$$

   It is important to note that the better is the solution, the greater is the increment of the pheromone.

ANT-CSP Algorithm 3/3

5. An *elitist strategy* is used to update pheromone trails:
   - pheromone trails increment is proportional to the distance of the current string from the input set, according to the rule:

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \left(1 - \frac{HD}{m}\right).$$

It is important to note that the better is the solution, the greater is the increment of the pheromone.

Ant-CSP Algorithm 3/3

5. An *elitist strategy* is used to update pheromone trails:
   - pheromone trails increment is proportional to the distance of the current string from the input set, according to the rule:

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \left(1 - \frac{HD}{m}\right).$$

It is important to note that the better is the solution, the greater is the increment of the pheromone.

Pseudocode 1/2

```
INITIALIZATION
while not (TERMINATION_CRITERION) do
    for i ← 1 to u do
        COLONY_i ← new_ant()
        COLONY_i.find_solution()
        COLONY_i.evaluate_solution()
    end for
    EVAPORATION
    COLONY_best.update_trails()
end while
```

Pseudocode 2/2

```
procedure INITIALIZATION
    for i ← 1 to m do
        for j ← 1 to |Σ| do
            T_ij ← 1/|Σ|
        end for
    end for
    initialize COLONY
end procedure
```

```
procedure EVAPORATION
    for i ← 1 to m do
        for j ← 1 to |Σ| do
            T_ij ← (1 − ρ) · T_ij;
        end for
    end for
end procedure
```

Ant-CSP, Simulated Annealing and Genetic Algorithm

We compared the Ant-CSP algorithm with two other approaches for the CSP problem [Liu *et al.*, 2005]:

- Simulated Annealing
- Genetic Algorithm

Ant-CSP, Simulated Annealing and Genetic Algorithm

We compared the Ant-CSP algorithm with two other approaches for the CSP problem [Liu *et al.*, 2005]:

- Simulated Annealing
- Genetic Algorithm

Ant-CSP, Simulated Annealing and Genetic Algorithm

We compared the Ant-CSP algorithm with two other approaches for the CSP problem [Liu *et al.*, 2005]:

- Simulated Annealing
- Genetic Algorithm

Simulated Annealing 1/4

Simulated Annealing (SA) is a generalization of Monte Carlo methods, originally proposed by [Metropolis *et al.*, 1953] as a means of finding the equilibrium configuration of a collection of atoms at a given temperature.

[Kirkpatrick *et al.*, 1983] first proposed to apply SA to solve combinatorial optimization problems.

The basic idea of SA was taken from an analogy with the annealing process used in metallurgy.

Simulated Annealing 1/4

Simulated Annealing (SA) is a generalization of Monte Carlo methods, originally proposed by [Metropolis *et al.*, 1953] as a means of finding the equilibrium configuration of a collection of atoms at a given temperature.

[Kirkpatrick *et al.*, 1983] first proposed to apply SA to solve combinatorial optimization problems.

The basic idea of SA was taken from an analogy with the annealing process used in metallurgy.

Simulated Annealing 1/4

Simulated Annealing (SA) is a generalization of Monte Carlo methods, originally proposed by [Metropolis *et al.*, 1953] as a means of finding the equilibrium configuration of a collection of atoms at a given temperature.

[Kirkpatrick *et al.*, 1983] first proposed to apply SA to solve combinatorial optimization problems.

The basic idea of SA was taken from an analogy with the annealing process used in metallurgy.

Simulated Annealing 2/4

The SA algorithm for the CSP problem by [Liu *et al.*, 2005] works much along the same lines as Kirkpatrick's algorithm:

1. the algorithm starts at temperature T, set to $m/2$, where m is the common string length.

Simulated Annealing 2/4

The SA algorithm for the CSP problem by [Liu *et al.*, 2005] works much along the same lines as Kirkpatrick's algorithm:

1. the algorithm starts at temperature T, set to $m/2$, where m is the common string length.

Simulated Annealing 3/4

2. For each temperature value, a block of L iterations is performed:
   - at each iteration, a new string u' of length m, over Σ, is constructed;
   - the energy change $\Delta E = H(u', S) - H(u, S)$ is evaluated, where S is the input set of strings;
   - if $\Delta E \leq 0$, u' becomes the new current solution, otherwise u' is chosen as current solution with a Boltzmann probability $e^{-\frac{\Delta E}{T}}$ only.

Simulated Annealing 3/4

2. For each temperature value, a block of L iterations is performed:
   - at each iteration, a new string u' of length m, over Σ, is constructed;
   - the energy change $\Delta E = H(u', S) - H(u, S)$ is evaluated, where S is the input set of strings;
   - if $\Delta E \leq 0$, u' becomes the new current solution, otherwise u' is chosen as current solution with a Boltzmann probability $e^{-\frac{\Delta E}{T}}$ only.

Simulated Annealing 3/4

2. For each temperature value, a block of L iterations is performed:
   - at each iteration, a new string u' of length m, over Σ, is constructed;
   - the energy change $\Delta E = H(u', S) - H(u, S)$ is evaluated, where S is the input set of strings;
   - if $\Delta E \leq 0$, u' becomes the new current solution, otherwise u' is chosen as current solution with a Boltzmann probability $e^{-\frac{\Delta E}{T}}$ only.

Simulated Annealing 3/4

2. For each temperature value, a block of L iterations is performed:
   - at each iteration, a new string u' of length m, over Σ, is constructed;
   - the energy change $\Delta E = H(u', S) - H(u, S)$ is evaluated, where S is the input set of strings;
   - if $\Delta E \leq 0$, u' becomes the new current solution, otherwise u' is chosen as current solution with a Boltzmann probability $e^{-\frac{\Delta E}{T}}$ only.

Simulated Annealing 4/4

3. At the end of each block of iterations, the temperature value is multiplied by a reduction factor γ.

The algorithm stops when a suitable termination criterion is met.

Simulated Annealing 4/4

3. At the end of each block of iterations, the temperature value is multiplied by a reduction factor γ.

The algorithm stops when a suitable termination criterion is met.

Genetic Algorithm 1/4

Genetic algorithms (GA) were first proposed by [Holland, 1975] as an abstraction of the biological evolution of living organisms.

GAs are based on natural selection and sexual reproduction processes.

Genetic Algorithm 1/4

Genetic algorithms (GA) were first proposed by [Holland, 1975] as an abstraction of the biological evolution of living organisms.
GAs are based on natural selection and sexual reproduction processes.

Genetic Algorithm 2/4

1. An initial population $P(t)$ of random candidate solutions $ind_0, ..., ind_{popsize-1}$ is generated:
   - each solution is a string of length m over the alphabet Σ;
   - each individual in the current population is evaluated by a *fitness* function $f = m - H_{max}$, where H_{max} is the maximum Hamming distance of s from all strings in S.

Genetic Algorithm 2/4

1. An initial population $P(t)$ of random candidate solutions $ind_0, \ldots, ind_{popsize-1}$ is generated:
   - each solution is a string of length m over the alphabet Σ;
   - each individual in the current population is evaluated by a *fitness* function $f = m - H_{max}$, where H_{max} is the maximum Hamming distance of s from all strings in S.

Genetic Algorithm 2/4

1. An initial population $P(t)$ of random candidate solutions $ind_0, ..., ind_{popsize-1}$ is generated:
    - each solution is a string of length m over the alphabet Σ;
    - each individual in the current population is evaluated by a *fitness* function $f = m - H_{max}$, where H_{max} is the maximum Hamming distance of s from all strings in S.

Genetic Algorithm 3/4

2. A crossover step allows to generate new individuals from members of the current population:
   - two "parental individuals" are randomly selected; then the crossover exchanges a randomly selected segment in this pair, so that two new strings are generated.

Genetic Algorithm 3/4

2. A crossover step allows to generate new individuals from members of the current population:
   - two "parental individuals" are randomly selected; then the crossover exchanges a randomly selected segment in this pair, so that two new strings are generated.

Genetic Algorithm 4/4

3. A mutation operator is applied to each individual:
   - it consists in exchanging two random positions in the string.
4. At this intermediate stage, there are two populations, namely, parents and offsprings. To create the next generation, an elitist strategy is applied.

Reproduction and mutation steps are repeated until a termination criterion is met.

Genetic Algorithm 4/4

3. A mutation operator is applied to each individual:
   - it consists in exchanging two random positions in the string.
4. At this intermediate stage, there are two populations, namely, parents and offsprings. To create the next generation, an elitist strategy is applied.

Reproduction and mutation steps are repeated until a termination criterion is met.

Genetic Algorithm 4/4

3. A mutation operator is applied to each individual:
   - it consists in exchanging two random positions in the string.
4. At this intermediate stage, there are two populations, namely, parents and offsprings. To create the next generation, an elitist strategy is applied.

Reproduction and mutation steps are repeated until a termination criterion is met.

Genetic Algorithm 4/4

3. A mutation operator is applied to each individual:
   - it consists in exchanging two random positions in the string.
4. At this intermediate stage, there are two populations, namely, parents and offsprings. To create the next generation, an elitist strategy is applied.

Reproduction and mutation steps are repeated until a termination criterion is met.

Experimental results

Experimental Protocol 1/3

- We have tested the SA-CSP, the GA-CSP, and the ANT-CSP algorithms using the azotated compounds alphabet $\Sigma = \{A, C, G, T\}$ of the fundamental components of nucleic acids.
- In our test platform, we considered a number of input strings $n \in \{10, 20, 30, 40, 50\}$, and string length $m \in \{10, 20, ..., 50\} \cup \{100, 200, ..., 1000\}$.

Experimental Protocol 1/3

- We have tested the SA-CSP, the GA-CSP, and the ANT-CSP algorithms using the azotated compounds alphabet $\Sigma = \{A, C, G, T\}$ of the fundamental components of nucleic acids.
- In our test platform, we considered a number of input strings $n \in \{10, 20, 30, 40, 50\}$, and string length $m \in \{10, 20, ..., 50\} \cup \{100, 200, ..., 1000\}$.

Experimental Protocol 2/3

- For each of a randomly generated problem instances, all algorithms were run 20 times.
- The total colony size for the ANT-CSP algorithm as well as the population size for the GA-CSP algorithm have been set to 10, whereas the number of generations has been set to 1,500. In the case of the SA-CSP algorithm, we fixed the number of function evaluations in 15,000.

Experimental Protocol 2/3

- For each of a randomly generated problem instances, all algorithms were run 20 times.
- The total colony size for the ANT-CSP algorithm as well as the population size for the GA-CSP algorithm have been set to 10, whereas the number of generations has been set to 1,500. In the case of the SA-CSP algorithm, we fixed the number of function evaluations in 15,000.

Experimental Protocol 3/3

Our tests have been performed on an *Intel Pentium M 750, 1.86 GHz, 1 GB RAM*, running *Ubuntu Linux*.

For each length, we computed the average *(AVG)* of the closest string scores *(HD)* found in the 20 runs and the standard deviation σ. Also, we computed the average of the running time *(Time)* (in milliseconds) over the 20 runs *(AVG)*.

Best results are reported in bold.

Experimental Protocol 3/3

Our tests have been performed on an *Intel Pentium M 750, 1.86 GHz, 1 GB RAM*, running *Ubuntu Linux*.
For each length, we computed the average *(AVG)* of the closest string scores *(HD)* found in the 20 runs and the standard deviation σ. Also, we computed the average of the running time *(Time)* (in milliseconds) over the 20 runs *(AVG)*.
Best results are reported in bold.

Experimental Results 1/5

Size (m)	SA-CSP			GA-CSP			Ant-CSP		
	HD		Time	HD		Time	HD		Time
	AVG	σ	AVG	AVG	σ	AVG	AVG	σ	AVG
10	8.45	0.497	67.5	**6.9**	0.3	1840	7.05	0.218	50.5
20	15.9	0.384	112	13.3	0.714	1860	**13.1**	0.589	97
30	23.6	0.663	216	19.6	0.583	2700	**19.3**	0.557	200
40	31.4	0.589	313	25.3	0.714	3040	**25.1**	0.654	281
50	38.8	0.678	428	31.8	0.994	3220	**31.6**	0.805	386
100	75.9	0.943	465	63.4	1.31	2060	**62.2**	0.766	433
200	151	1.04	901	129	1.43	2290	**124**	1.58	855
300	226	1.18	1350	195	2.19	2540	**188**	1.57	1290
400	301	2.01	1780	262	2.52	2720	**252**	1.68	1700
500	375	2.05	2190	330	2.52	2940	**317**	2.15	2110
600	450	1.87	2740	400	3.71	3800	**385**	2.5	2920
700	525	1.68	3980	470	3.43	4860	**451**	2.95	4270
800	600	1.51	3720	540	4.04	4370	**517**	2.11	3860
900	675	1.19	5670	610	4.01	6110	**585**	4.05	5690
1000	750	1.53	7720	680	4.12	7850	**652**	3.72	7850

Table: Results for inputset of 10 strings of length m.

Experimental Results 2/5

Size (m)	SA-CSP			GA-CSP			Ant-CSP		
	HD		Time	HD		Time	HD		Time
	AVG	σ	AVG	AVG	σ	AVG	AVG	σ	AVG
10	8.95	0.384	211	**7.95**	0.218	3560	**7.95**	0.218	132
20	17.1	0.589	342	**14.8**	0.4	3460	**14.8**	0.4	258
30	24.8	0.536	502	21.6	0.497	3300	**21.4**	0.49	370
40	32.5	0.497	602	28.1	0.477	3220	**28**	0.632	452
50	40.1	0.726	735	35	0.589	3300	**34.8**	0.536	546
100	78.4	0.663	874	69.5	0.921	2250	**67.7**	0.853	646
200	154	0.917	2070	140	1.74	3370	**135**	0.963	1460
300	229	1.16	2300	210	2.09	2970	**203**	1.95	1810
400	305	1.18	4460	281	1.95	4980	**272**	1.56	3090
500	380	1.25	5270	353	2.52	4930	**341**	1.65	3510
600	456	1.46	4610	426	1.89	4180	**411**	1.68	3660
700	531	1.16	6280	499	3.51	4770	**482**	1.95	4350
800	607	1.32	11300	572	1.88	9370	**553**	2.84	7780
900	682	1.49	13700	645	2.58	10800	**623**	2.51	10400
1000	757	1.69	15700	720	2.79	11800	**695**	2.49	11800

Table: Results for inputset of 20 strings of length m.

Experimental Results 3/5

Size (m)	SA-CSP			GA-CSP			Ant-CSP		
	HD		Time	HD		Time	HD		Time
	AVG	σ	AVG	AVG	σ	AVG	AVG	σ	AVG
10	9	0	245	8.25	0.433	2830	**8.15**	0.357	148
20	17.3	0.458	518	15.3	0.458	3460	**15.2**	0.4	341
30	25.1	0.357	772	22.7	0.458	3520	**22.4**	0.477	508
40	33	0.316	985	29.5	0.5	3720	**29.1**	0.357	638
50	40.9	0.539	1230	36.9	0.357	4180	**36.1**	0.436	814
100	79.3	0.557	1280	72.2	0.726	2450	**70.8**	0.536	850
200	156	0.829	4760	144	1.08	5800	**140**	0.975	2750
300	232	0.831	6640	216	1.77	6610	**209**	1.27	4260
400	308	0.829	9160	290	2.93	8160	**280**	1.28	5550
500	383	0.963	11110	362	1.66	8830	**351**	1.79	6760
600	459	1.24	12500	436	2.14	9800	**423**	1.95	7610
700	534	1.03	14500	510	2.57	10900	**495**	2.01	9430
800	610	1.14	17700	583	2.57	12600	**568**	2.36	10300
900	686	1.69	19800	658	3.42	13200	**640**	2.09	11400
1000	760	2.24	19800	731	2.97	12400	**713**	2.29	10700

Table: Results for inputset of 30 strings of length m.

Experimental Results 4/5

Size (m)	SA-CSP			GA-CSP			Ant-CSP		
	HD		Time	HD		Time	HD		Time
	AVG	σ	AVG	AVG	σ	AVG	AVG	σ	AVG
10	9.4	0.49	428	8.9	0.3	4000	**8.55**	0.497	252
20	17.6	0.477	742	15.9	0.218	3990	**15.8**	0.433	471
30	25.6	0.49	1210	23.1	0.384	4690	**22.9**	0.384	754
40	33.3	0.458	1540	30.4	0.572	4640	**30.1**	0.218	962
50	41.2	0.433	1940	37.5	0.497	5070	**37**	0.589	1220
100	80	0.669	2080	73.6	0.663	3420	**71.7**	0.477	1260
200	157	0.889	5740	146	1.24	5570	**142**	0.669	3230
300	233	0.889	8760	219	0.954	8640	**214**	1.05	5550
400	309	0.831	10090	293	1.87	9510	**285**	1.16	6560
500	385	0.748	14800	368	2.07	11000	**358**	1.24	7330
600	461	1.01	17800	441	1.69	13100	**431**	1.91	7940
700	536	1.05	21700	515	2.1	14300	**503**	1.01	11700
800	612	1.1	23500	590	2.34	14300	**577**	1.93	11300
900	688	1.34	26700	664	2.52	17200	**649**	2.31	15600
1000	763	1.43	30900	738	2.62	15900	**722**	1.91	16000

Table: Results for inputset of 40 strings of length m.

Experimental Results 5/5

Size (m)	SA-CSP			GA-CSP			Ant-CSP		
	HD		Time	HD		Time	HD		Time
	AVG	σ	AVG	AVG	σ	AVG	AVG	σ	AVG
10	9.45	0.497	574	9	0	4390	**8.85**	0.357	334
20	17.8	0.433	1030	16.2	0.4	4620	**16.1**	0.218	620
30	25.9	0.3	1490	23.5	0.5	4820	**23.2**	0.4	899
40	33.5	0.497	1960	30.9	0.357	5070	**30.6**	0.497	1180
50	41.7	0.458	2410	38.2	0.433	5270	**37.8**	0.433	1450
100	80.6	0.49	2970	74.7	0.64	3970	**73.3**	0.64	1750
200	158	0.671	9090	148	0.91	8530	**144**	0.698	5550
300	234	0.678	14000	222	0.91	10900	**216**	0.889	8320
400	310	0.792	18500	297	1.65	13100	**289**	1.41	11100
500	386	1.16	21900	369	1.69	14800	**362**	1.24	12900
600	462	1.13	21200	444	1.5	14500	**434**	1.74	12200
700	538	1.14	26800	519	1.9	17300	**508**	1.7	15500
800	614	1.43	28900	594	2.9	14000	**582**	2.29	13900
900	689	1.1	33500	667	1.64	19700	**656**	2.11	18800
1000	765	1.19	36600	742	3.09	21000	**729**	1.68	18300

Table: Results for inputset of 50 strings of length m.

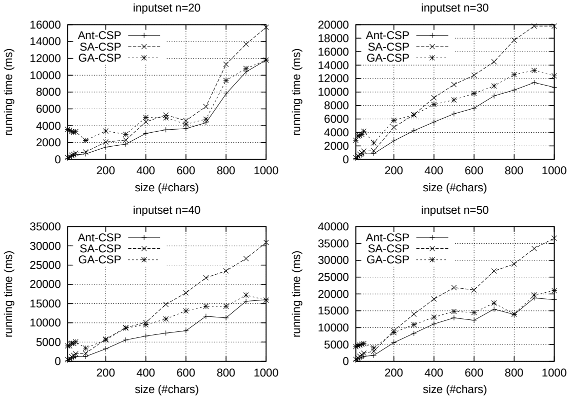

Figure: Running times plots for $n = 20, 30, 40, 50$. Notice that, as n increases, the gap between ANT-CSP and the other two algorithms becomes more noticeable.

Conclusions 1/2

- Experimental results show that the Ant-CSP always outperforms both the GA-CSP and the SA-CSP algorithms both in terms of solution quality and efficiency.
  In particular, in the case of short instances, i.e. for $10 \leq m \leq 50$, the Ant-CSP algorithm is from 5 to 36 times faster than GA-CSP.
- Furthermore, it turns out that as n increases, the gap between the running time of the Ant-CSP and the SA-CSP algorithms becomes considerable.

Conclusions 1/2

- Experimental results show that the ANT-CSP always outperforms both the GA-CSP and the SA-CSP algorithms both in terms of solution quality and efficiency.
  In particular, in the case of short instances, i.e. for $10 \leq m \leq 50$, the ANT-CSP algorithm is from 5 to 36 times faster than GA-CSP.
- Furthermore, it turns out that as n increases, the gap between the running time of the ANT-CSP and the SA-CSP algorithms becomes considerable.

Conclusions 2/2

- We also remark that the ANT-CSP provides results of a better quality than the other two algorithms in terms of Hamming distance.
- Finally we note that the ANT-CSP algorithm is quite robust, as its standard deviation σ remains low.

Conclusions 2/2

- We also remark that the ANT-CSP provides results of a better quality than the other two algorithms in terms of Hamming distance.
- Finally we note that the ANT-CSP algorithm is quite robust, as its standard deviation σ remains low.

Future works

Future works will be focused on two fronts:

- performance improvements;
- search for heuristic information to improve quality of solutions and convergence speeds.

Additionally, we plan to extend our algorithm to the Closest Substring Problem.

Future works

Future works will be focused on two fronts:

- performance improvements;
- search for heuristic information to improve quality of solutions and convergence speeds.

Additionally, we plan to extend our algorithm to the Closest Substring Problem.

Future works

Future works will be focused on two fronts:

- performance improvements;
- search for heuristic information to improve quality of solutions and convergence speeds.

Additionally, we plan to extend our algorithm to the Closest Substring Problem.

Future works

Future works will be focused on two fronts:

- performance improvements;
- search for heuristic information to improve quality of solutions and convergence speeds.

Additionally, we plan to extend our algorithm to the Closest Substring Problem.

Deneubourg, J., Aron, S., Goss, S., & Pasteels, J. (1990).
Journal of Insect Behavior, **3** (2), 159–168.

Dorigo, M. (1992).
Optimization, Learning and Natural Algorithms.
PhD thesis Dipartimento di Elettronica, Politecnico di Milano, Italy.

Dorigo, M., Caro, G., & Gambardella, L. (1999).
Artificial Life, **5** (2), 137–172.

Frances, M. & Litman, A. (1997).
Theory of Computing Systems, **30** (2), 113–119.

Gasieniec, L., Jansson, J., & Lingas, A. (1999).
In: *Proceedings of the 10th annual ACM-SIAM Symposium on Discrete algorithms* pp. 905–906, Society for Industrial and Applied Mathematics Philadelphia, PA, USA.

Hertz, G., Hartzell, G., & Stormo, G. (1990).
Bioinformatics, **6** (2), 81–92.

Holland, J. (1975).
Adaptation in Natural and Artificial Systems.
The MIT Press.

Kirkpatrick, S., Gelatt, C., & Vecchi, M. (1983).
Science, **220** (4598), 671–680.

Lanctot, J., Li, M., Ma, B., Wang, S., & Zhang, L. (1999).
In: *Proceedings of the 10th annual ACM-SIAM Symposium on Discrete algorithms* pp. 633–642, Society for Industrial and Applied Mathematics Philadelphia, PA, USA.

Liu, X., He, H., & Sykora, O. (2005).
In: *Advanced Data Mining and Applications* volume 3584 of *Lecture Notes in Computer Science* pp. 591–597, Springer Berlin/Heidelberg.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., & Teller, E. (1953).
J. Chem. Phys, **21**, 1087–1092.

Roman, S. (1992).

Coding and information theory.
Springer.

Stormo, G. & Hartzell, G. (1989).
In: *Proc. Natl. Acad. Sci. USA* volume 86 pp. 1183–1187,.