Soft Computing (2022) 26:5765–5772
https://doi.org/10.1007/s00500-022-06969-1

OPTIMIZATION

Parallel *m*-dimensional relative ant colony optimization (*m*DRACO) for the Costas-array problem

David Vulakh[1] · **Raphael Finkel**[2]

Accepted: 23 February 2022 / Published online: 26 March 2022

Abstract
The Costas-array problem is a combinatorial constraint-satisfaction problem (CSP) that remains unsolved for many array sizes greater than 30. In order to reduce the time required to solve large instances, we present an Ant Colony Optimization algorithm called *m*-Dimensional Relative Ant Colony Optimization (*m*DRACO) for combinatorial CSPs, focusing specifically on the Costas-array problem. This paper introduces the optimizations included in *m*DRACO, such as map-based association of pheromone with arbitrary-length component sequences and relative path storage. We assess the quality of the resulting *m*DRACO framework on the Costas-array problem by computing the efficiency of its processor utilization and comparing its run time to that of an ACO framework without the new optimizations. *m*DRACO gives promising results; it has efficiency greater than 0.5 and reduces time-to-first-solution for the $m = 16$ Costas-array problem by a factor of over 300.

Keywords CSP · Costas array · ACO · Parallelism

1 Introduction

Costas arrays are square arrays of 1s and 0s such that there is exactly one element with value 1 in each row and column and no three or four distinct elements with value 1 form a (potentially degenerate) parallelogram (Drakakis 2011). John P. Costas first described these arrays in 1965 as representations of sequences of SONAR pulse frequencies with optimal auto-correlation properties (Costas 1984); they have since been applied in other areas, such as communication systems and cryptography (Taylor et al. 2011). Costas arrays are also objects of theoretical interest to the research community. Golomb and Taylor (1984) describe several algebraic constructions for certain classes of Costas arrays in 1984, which remain the only known Costas-array constructions (Drakakis 2011). All Costas arrays of order $m \leq 29$ are known (Drakakis et al. 2011), but the existence of Costas arrays for many sizes $m \geq 30$ remains an open question (Drakakis 2011).

The problem of finding Costas arrays is a constraint-satisfaction problem (CSP). CSPs are multi-variable combinatorial problems that require that each variable be assigned a value such that the assignments satisfy a set of constraints. Because there exist *NP*-complete CSP instances (Bulatov et al. 2000), no polynomial-time algorithm for the general CSP is known. In order to reduce the time to solve large CSP instances, researchers have developed parallel algorithms to pool the computational resources of multiple processors. The Costas-array problem is a common benchmark problem for assessing parallelized CSP solvers (see Arbelaez and Codognet 2014, Caniou et al. 2015, and Truchet et al. 2013 for examples).

Researchers also apply heuristic algorithms that perform stochastic or approximate searches to combinatorial problems. One family of such algorithms is the Ant Colony Optimization (ACO) metaheuristic. Modeled after the foraging patterns of ants, ACO algorithms traverse a graph of the search space, probabilistically preferring paths with components of higher learned favorability (Dorigo and Di Caro 1999). Dorigo proposed the first ACO algorithm (Dorigo et al. 1996) to solve constraint-optimization problems such as the Traveling Salesman Problem (TSP), but researchers have applied ACO to CSPs as well (Roli et al. 2001).

✉ David Vulakh
dvulakh@mit.edu

Raphael Finkel
raphael@cs.uky.edu

[1] Math, Science, and Technology Center, Paul Laurence Dunbar High School, Lexington, KY, USA

[2] Computer Science Department, University of Kentucky, Lexington, KY, USA

1.1 Costas arrays

Definition 1 A *constraint-satisfaction problem (CSP)* (Bulatov et al. 2000) is an ordered triple (V, D, C), where V is the set of *variables*, D is the *domain*,[1] and C is the set of *constraints*. Each constraint C_i is of the form (s_i, ρ_i), where s_i describes a tuple of variables $s_i : \{1, \ldots, m_i\} \to V$ and ρ_i is an m_i-ary relation on D, $\rho_i \subseteq D^{m_i}$.

A *solution* of the constraint-satisfaction problem instance (V, D, C) is a function $f : V \to D$ such that for all constraints $C_i = (s_i, \rho_i)$, $f \circ s_i \in \rho_i$.

In general, each variable V_i of a CSP can have its own corresponding domain D_i. Rather than explicitly defining a domain D_i for each variable in the definition above, we let $D = \bigcup D_i$ and add a unary constraint (s, ρ) such that $s(1) = V_i$ and $\rho = D_i$ to each variable.

Definition 2 A *Costas array* (Drakakis 2011) of size m is an $m \times m$ array of 1s and 0s such that there is exactly one element with value 1 in each row and column, and there are no equal displacement vectors between distinct pairs of distinct elements with value 1. (Equivalently, no three or four elements with value 1 form a (potentially degenerate) parallelogram.)

Costas arrays are permutation arrays, so each Costas array A of size m can be represented as a permutation P of the integers from 1 through m such that, if $A_{ij} = 1$, then $P_i = j$. A permutation that corresponds to a Costas array in this manner is said to satisfy the Costas-array property.

The Costas-array problem is an existence problem. Whether there exist Costas arrays of size m for all positive integers m remains an open question (Drakakis 2011). The smallest m for which it is unknown whether a Costas array of size m exists is 32 (Drakakis 2011).

1.2 Ant colony optimization

1.2.1 Summary of the ACO metaheuristic

Ant Colony Optimization (ACO) is a class of heuristics for solving constraint-optimization problems.[2] In ACO algorithms, processes called ants probabilistically traverse a graph representing the search space, modifying the graph properties that determine ant behavior to increase the probability of future ants selecting paths with optimal properties. We summarize the elements of an ACO algorithm and its ant processes as described in Dorigo and Di Caro (1999).

- $C = \{c_1, \ldots, c_n\}$ is a finite set of *components*
- $L = \{l_{ij} : (c_i, c_j) \in \tilde{C}\}$, where $\tilde{C} \subseteq C^2$ is a set of *connections* between components
- Any finite sequence of components $s = (c_i, \ldots, c_j)$ is called a *state*; states need not be of a particular length or contain all components
- S is the set of all possible states
- J_s is a *cost* associated with the state s
- Ω is a set of *constraints* on S
- P is a set of *requirements* on S
- $\tilde{S} \subseteq S$ is the set of all states that satisfy the constraints Ω; $\tilde{S}$ is then the set of *feasible states*
- $\psi \in \tilde{S}$ is a *solution* if ψ satisfies all requirements P
- Given two states $s_1 = (c_i, \ldots, c_j), s_2 \in S$, s_2 is a *neighbor* of s_1 if $s_2 \in \tilde{S}$ and there exists c_k such that $(c_i, \ldots, c_j, c_k) = s_2$ (that is, s_1 is a prefix of s_2); the set $\mathcal{N}_s$ of states that are neighbors of s is called the *neighborhood* of s.

The graph $G = (C, L)$ is the *construction graph*. Any solution ψ can be expressed as a feasible path on G; we seek a solution ψ' such that $J_{\psi'}$ is minimal.

Information is associated with states s (paths on G) and $i \in \mathcal{N}_s$ (possible next steps) in the form of *pheromone values* τ_{si} and *heuristic values* η_{si}. Ants, which search for the solution ψ', have the following properties:

- An ant k can be assigned a start state s_k and one or more end conditions e_k
- An ant k in state s can move to any component such that the resulting state $s' \in \mathcal{N}_s$
- The probability $\mathcal{P}_{si}$ that an ant k in state s moves to i is governed by a function of τ_{si} and η_{si} called the *ant-routing table* $\mathcal{A}_{si}$
- An ant k *dies* once it reaches an end state e_k
- If ant k has started at state s_k and after a sequence of steps has died, it has completed one *iteration*.

ACO also executes procedures outside of the ant processes. *Pheromone evaporation* is the periodic decrease in all pheromone values to prevent early convergence on a suboptimal solution. Based on properties of the solutions discovered by ants, ACO also increases some pheromone values to encourage exploration of promising states. Ants indirectly communicate with each other to improve solution quality through these changes to the pheromone values.

1.2.2 Prior work

ACO algorithms have been adapted to efficiently solve optimization and constraint problems in a variety of real-world and theoretical settings including: job scheduling in mining supply chains (Thiruvady et al. 2016); planning selective maintenance (Liu et al. 2018); vehicle routing (Zhang et al.

[1] In this paper, we only consider constraint-satisfaction problems with domains of finite cardinality.

[2] We describe the adaptation of ACO to constraint-satisfaction problems in Sect. 2.1.

2019; Yan 2018); k-SAT (Ning et al. 2018); n-Queens Completion (Ye et al. 2017); and Minimum Connected Dominating Set (Bouamama et al. 2019). ACO effectively solves optimization problems even in probabilistic settings, where the computation of the heuristic value η can be more complicated than in simpler problems like TSP (Liu et al. 2018).

The main deficiency of ant colony algorithms in general is the tendency to prematurely converge on local minima that are globally suboptimal (Deng et al. 2019; Guan et al. 2019); many successful implementations of ACO achieve performance improvements by adding countermeasures against early convergence. To this end, many ACO implementations are augmented with an additional non-ACO optimization step, such as Local Search (LS) (Guan et al. 2021, 2019; Thiruvady et al. 2016). Performing LS on the solutions found by an ant colony can both improve convergence time by examining promising regions of the space more systematically and prevent early stagnation by sufficiently perturbing solutions that have reached a local minimum. For example, Guan et al. (2021) implement an automatic updating mechanism within ACO (AU-ACO) that attempts to replace the solution constructed by each ant with a better one that differs by only one variable assignment; the resulting AU-ACO algorithm converges approximately five to ten times faster than ACO without automatic updating on binary CSP tests. Some implementations apply a more complicated LS step less frequently, such as only once the ant colony detects that it is in a stagnant state; for example, Guan et al. (2019) maintain the information entropy associated with each path, initiating a crossover-based local search once some iteration adds little information to the system. (The entropy along the best-discovered path does not change significantly.) In addition to LS, Thiruvady et al. (2016) consider augmenting ACO with simulated annealing, but find this approach ineffective. Others (Bouamama et al. 2019; Zhang et al. 2019) likewise apply additional mutations to discovered solutions when adapting ACO to the Minimum Connected Dominating Set and Vehicle Routing problems, respectively.

A second approach for improving ACO implementations focuses on modifying the interactions between ants by considering novel methods of manipulating the application of pheromone to the problem graph. Ye et al. (2017) augment the path-construction process with a system of pheromone-based negative feedback (in addition to the standard positive feedback of ACO) to steer ants away from explored poor solutions; this ACO with negative feedback (ACON) significantly outperforms their implementation of standard ACO. Ning et al. (2018) monitor the degree of similarity of solutions found in consecutive rounds and employs a pheromone-smoothing technique to escape local minima when stagnation is detected, achieving superior performance. Deng et al. (2019) likewise consider a rule for diffusing pheromone applied by ants to nearby states to further encourage exploration and prevent stagnation.

1.2.3 Our contributions

We present a novel ACO algorithm, m-Dimensional Relative Ant Colony Optimization (mDRACO). The central idea behind mDRACO is to improve ACO performance by maximizing the information contained in the pheromone through which ants interact with each other; this idea contrasts with approaches that augment ACO with LS or manipulate pheromone primarily with the objective of reducing the chance of stagnation.

As discussed in Sect. 2.4.1, mDRACO associates pher-o-mone with arbitrary-length subsequences of components, so the pheromone table has m dimensions (where m is the number of components). Typically, ACO only associates pheromone with the favorability of transitioning from one component to another, which produces a two-dimensional table. mDRACO also associates pheromone information in a *relative* manner, as described in Sect. 2.4.2. These optimizations work best when coupled with a branch-and-bound heuristic for the path-selection process, as discussed in Sect. 2.4.3. Our ACO implementation performs substantially better with the mDRACO optimizations, reducing median run time for the Costas-array problem at $m = 16$ by a factor of over 300.

2 Algorithm implementation

2.1 Adaptation of ACO to CSPs

We apply ACO algorithms to constraint-satisfaction problems as described in Roli et al. (2001) by treating CSPs as maximal-constraint-satisfaction problems, which are a class of constraint-optimization problems. To solve the CSP (V, D, C), we let each of the components (the vertices of the graph G) represent the assignment of a single value from D to a single variable in V. The constraints C are partitioned into hard constraints Ω and soft constraints ω. Ω defines the set of feasible states $\tilde{S}$ visitable by ants, and ω defines the cost of these states, so that for $s \in \tilde{S}$, J_s is the number of violations of the constraints ω in s.

Since ants seek the solution ψ' that has minimal cost, they seek to minimize the number of constraint violations. If $J_{\psi'} = 0$ for some ψ', then ψ' contains no violations of the constraints in $\Omega \cup \omega = C$ and therefore represents a solution to the original constraint-satisfaction problem (V, D, C). In order to apply ACO to strongly constrained CSPs effectively, C must be partitioned so that Ω is not so strongly constrained that a state $s \in \tilde{S}$ not representing a solution ψ has $|\mathcal{N}_s| = 0$; if the construction of feasible paths by ants is intractable,

then pheromone deposition does not occur and ACO cannot exhibit its heuristic learning behavior.

2.2 Application to Costas arrays

We use an ACO algorithm called the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) (Stützle and Hoos 2000) as a starting point for our implementation, modeling the values τ_{si}, η_{si}, and J_s after the standard ACO TSP solution (Dorigo and Di Caro 1999). In $\mathcal{MM}$AS, all pheromone values τ_{si} satisfy $\tau_{\min} \leq \tau_{si} \leq \tau_{\max}$ for some constants $\tau_{\min}$ and $\tau_{\max}$. When searching for Costas arrays of size m, we use $\{1, \ldots, m\}$ for the set of components, so that states (paths in G) represent sequences of integers. The sole hard constraint in Ω is that no state $s \in \tilde{S}$ contains the same component twice; the sole requirement in P is that solutions ψ must have length m. Any solution ψ constructed by the ants is therefore a Hamiltonian tour of G and represents a permutation of the integers from 1 through m.

The cost J_s of a state $s \in \tilde{S}$ is the number of Costas-array-property violations in the permutation it represents. The heuristic value η_{si} is one divided by the additional cost incurred by moving to i from s,

$$\eta_{si} = \begin{cases} \dfrac{1}{J_i - J_s} & J_i - J_s > 0 \\ \eta_{\max} & J_i - J_s = 0 \end{cases} \tag{1}$$

Here, $\eta_{\max}$ is a large constant. (In our implementation, we set $\eta_{\max} = 2{,}000{,}000{,}000$.) When $J_i = J_s$, no new constraint violations are incurred by moving from J_s to J_i, and any such state J_i should be strongly preferred over one that introduces new constraint violations.

We describe the computation of τ_{si} and $\mathcal{A}_{si}$ in Sects. 2.3 (Eq. 3) and 2.4.1 (Eq. 4). $\mathcal{P}_{si}$ is the probability of selecting i from $\mathcal{N}_s$ if each element i of $\mathcal{N}_s$ has weight $\mathcal{A}_{si}$,

$$\mathcal{P}_{si} = \frac{\mathcal{A}_{si}}{\sum_{j \in \mathcal{N}_s} \mathcal{A}_{sj}} \tag{2}$$

2.3 Standard pheromone association

Traditionally in ACO TSP Solvers, the pheromone value τ_{si} is associated with the edge in G from the last element of s to the last element in i (Stützle and Hoos 2000), that is, pheromone information represents the learned favorability of following one component with another. When the ant system deposits pheromone on a solution ψ, it increments τ_{si} by $\frac{1}{J_\psi}$ for all s, i such that the last element of i follows the last element of s in ψ and $i \in \mathcal{N}_s$.

The ant-routing table $\mathcal{A}_{si}$ for an ant in some state s determining the favorability of moving to state i is

$$\mathcal{A}_{si} = \begin{cases} 0 & i \notin \mathcal{N}_s \\ (\tau_{si})^\alpha \, (\eta_{si})^\beta & i \in \mathcal{N}_s \end{cases} \tag{3}$$

where α and β are constants that determine the relative weight of pheromone and heuristics values, respectively, in the ant decision-making process.

Pheromone evaporation occurs after each ant has completed one iteration. For all states $s \in \tilde{S}$, $i \in \mathcal{N}_s$, if the pheromone value representing the learned favorability of moving to i from s is τ_{si} before evaporation, it becomes $\tau'_{si} = \max\{(1-\rho)\tau_{si}, \tau_{\min}\}$ for a constant $0 \leq \rho < 1$ after.

2.4 Optimizations in the *m*DRACO algorithm

2.4.1 Arbitrary-length pheromone association

We attempt to increase the value of pheromone information by associating it with arbitrary-length sequences of components. To this end, we represent the set of all pheromone values deposited by ants as a map $T : \tilde{S} \to \mathbb{R}$ (implemented in our program as a multiple-reader single-writer default-value hash table). When our ant system deposits pheromone on a solution ψ, it increments $T(s)$ by $\frac{1}{J_\psi}$ for all s such that s is a contiguous subsequence of ψ of length greater than 1. If $s = (c_i, \ldots, c_j, c_k)$, then the value $T(s)$ now represents the learned favorably of following an arbitrary-length sequence of components $(c_i, \ldots, c_j)$ with c_k. During evaporation, we let each map entry $T(s)$ assume the new value $\max\{(1-\rho)T(s), \tau_{\min}\}$.

We now define the pheromone information associated with moving to i from s as two components: τ_{si} (pheromone value) and λ_{si} (trail length value). If τ_0 is the default map value[3] and j is the longest non-empty suffix of i for which $T(j) \neq \tau_0$, then $\tau_{si} = T(j)$ and λ_{si} is the length of j. If there is no such suffix, then $\tau_{si} = \tau_0$ and $\lambda_{si} = 1$.

We define the ant-routing table $\mathcal{A}_{si}$ for an ant in state s determining the favorability of moving to state i:

$$\mathcal{A}_{si} = \begin{cases} 0 & i \notin \mathcal{N}_s \\ (\tau_{si})^\alpha \, (\eta_{si})^\beta \, (\lambda_{si})^\gamma & i \in \mathcal{N}_s \end{cases} \tag{4}$$

[3] In accordance with the standard $\mathcal{MM}$AS algorithm, we initialize the default map value τ_0 to $\tau_{\max}$ and let this value evaporate after each iteration as though it were a map entry. At some point early in the search, τ_0 assumes a permanent value of $\tau_{\min}$. To keep map memory consumption from growing without bound over the course of the program's execution, we remove entries that, as a result of pheromone evaporation, map to $\tau_{\min}$, which should be positive.

where α, β, and γ are constants that determine the relative weight of pheromone, heuristic, and trail-length values, respectively, in the ant decision-making process.

We expect that longer pheromone trails carry more meaningful information, which is why we choose always to use the value in T associated with the longest suffix of i when determining τ_{si}. However, the value mapped by T to a long pheromone trail j is less than or equal to the value mapped by T to all suffixes of j, since a greater number of paths constructed by ants share shorter subsequences than longer ones. To account for this imbalance, we add the $(\lambda_{si})^{\gamma}$ factor in the ant-routing table entry.

2.4.2 Relative path storage

Because we expect two sequences for which the difference between corresponding elements is constant (e.g., (3, 1, 2, 5) and (4, 2, 3, 6)) to have related favorability in constructing Costas arrays, we further modify the process of pheromone association by using a canonical version of each state. If s is a sequence of integers with first element s_0, let the canonical sequence of s z^s be the sequence with the same length as s such that if the ith element of s is s_i, the ith element of z^s is $z^s_i = s_i - s_0$. Now, whenever looking up or modifying the value that T maps to a sequence s, we use $T(z^s)$. The result is that an ant considering the learned favorability of (3, 1, 2, 5) pools information collected about all s such that $z^s = z^{(3,1,2,5)} = (0, -2, -1, 2)$, including (4, 2, 3, 6). We do not expect this heuristic to improve mDRACO run time for combinatorial problems in general, since the numeric values assigned to domain elements are arbitrary for problems like the TSP. However, we do expect this optimization to generalize to other problems with costs based on relative positions of domain values, such as the N-Queens Problem.

2.4.3 Quality threshold

Some ants may, during the path construction process, arrive at a state s such that J_s is high and s is still far from being a solution (that is, s is short). Ants in such a position waste run time and pheromone map queries in finishing the construction of s, since the cost of any solution ψ with s as a prefix is necessarily at least J_s. To reduce resources wasted in this way, we apply a branch-and-bound heuristic by setting a quality threshold ϑ for the ants such that if ψ' is the lowest-cost solution found by the ant system and an ant is located at state s with $J_s > J_{\psi'} + \vartheta$, the ant dies without finishing the path s and creates a new path when respawned.

3 Assessment

We implement the ant system described in the C++ language using the pthread and *latomic* implementations of STL headers `<thread>` and `<atomic>`. We conduct our assessments by running this implementation on a Xeon Phi 7210 processor (Knight's Landing) with 64 cores supporting 256 threads, provided by the University of Kentucky Center for Computational Sciences. The machine runs CentOS 8. Throughout the experimentation process, we saturate the logical processors of the machine, employing approximately 250 threads at a time.

3.1 Efficiency

A standard metric of parallel-program performance is efficiency, a measure of the amount of parallel computational resources effectively applied to the solution; as the number of workers increases, the efficiency tends to decrease. The definition below describes specifically how we computed the values shown in Fig. 1.

Definition 3 (*Efficiency*) If a parallel ant system with a single worker solves a problem with median time to first solution T_0, and an execution of a parallel ant system with N workers finds a (possibly different) first solution to that problem in time T, then the *speedup* S and *efficiency* E of the N-worker system for that execution are

$$S = \frac{T_0}{T} \qquad E = \frac{S}{N} = \frac{T_0}{N \cdot T} \tag{5}$$

The definitions we give differ slightly from the traditional ones in sources such as (Cormen et al. 2009) on several points. First, since the mDRACO program is stochastic and the time to first solution varies across runs, we define speedup and efficiency to be properties of a single execution of the ant system rather than ones of the program as a whole. Second, while traditional definitions compare the run time of a parallel program running on N processors to that of its serialization, we choose to use a parallel ant system with one worker (and therefore two total processes) to determine T_0, because serializing our parallel ant system would require a change in the methods of communication between workers and the queen.

The efficiency results shown in Fig. 1 show $E > 0.5$ for $m \geq 14$. E decreases monotonically as N increases. The values of $E > 1$ are due to the stochastic nature of the ant system; some runs of the algorithm are lucky, leading to a quick discovery of a solution. The consistent value of $E \approx 0.6$ for $m \geq 14$ and $N \geq 200$ implies that the efficiency is likely to remain greater than 0.5 for larger problem sizes m and worker counts N. In our measure of efficiency, N is the number of workers utilized by the program. Even the $N = 1$

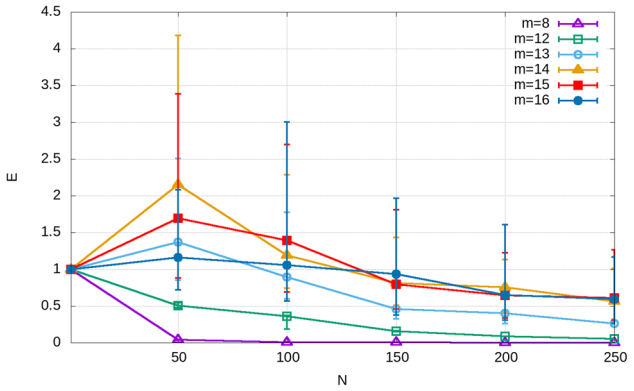

Fig. 1 Median E measured as a ratio of times to first solution (over 100 runs) for each number of workers N and problem size m. Error bars at the first and third quartiles

case has two processors available to it, so we expect optimal efficiency to be closer to 0.5 than to 1.

3.2 Optimized performance

To assess the quality of the three optimizations: arbitrary-length pheromone association, relative path storage, and the quality threshold ϑ, we compare the median time-to-first-solution over 100 runs of our implementation of mDRACO with various combinations of these optimizations. In all cases, the parameters α, β, γ, $\tau_{\min}$, and $\tau_{\max}$ are held constant. All constants are set to experimentally determined optimal values.

Figure 2 shows the median time-to-first-solution in seconds for problem sizes $12 \leq m \leq 19$, and Fig. 3 visualizes these data over $12 \leq m \leq 16$. Weaker combinations of settings take too long to complete for data collection to be reasonable; for example, out of a small sample of 2,0,NR runs on the size-17 Costas-array problem, only one finished within 50,000 seconds. For this reason, some table entries are left empty.

Though each of the optimizations is fairly weak on its own, the experimental results show that they are very powerful when applied together, and their usefulness appears to increase rapidly with problem size. Run-time differences are generally negligible for $m < 15$, and the slightly poorer performance of some of the optimized ACO variants is likely a result of overhead that does not pay off on easier problem instances, where solutions can often be found without much pheromone guidance. For $m \geq 15$, the differences in run times between different settings of combinations become pronounced. At $m = 15$, mDRACO (the rightmost column of Fig. 2) finds solutions approximately 50 times faster than our standard ACO implementation; at $m = 16$, mDRACO is approximately 300 times faster. We do not have a reliable estimate of the median time-to-first solution of standard ACO

m	2,0,NR	0,0,NR	2,20,NR	2,0,R	2,20,R	0,20,R
12	0.11	0.11	0.11	0.11	0.11	0.11
13	0.11	0.11	0.11	0.11	0.11	0.11
14	0.13	0.15	0.13	0.20	0.12	0.21
15	26.75	168.60	0.52	15.55	0.66	0.57
16	910.36	3628.63	789.08	472.17	8.45	2.92
17	—	—	—	—	132.98	17.75
18	—	—	—	—	9147.29	95.99
19	—	—	—	—	—	798.75

Fig. 2 Median time-to-first solution in seconds (over 100 runs) for our implementation of mDRACO with various combinations of optimizations enabled. The columns are labeled by triples λ, ϑ, r. $\lambda = 2$ indicates that pheromone is only associated with the transition from one state to the next (the standard ACO behavior), while $\lambda = 0$ indicates that pheromone is associated with arbitrary-length sequences of components (mDRACO behavior). $\vartheta = 0$ indicates that the branch-and-bound heuristic is not used (standard ACO behavior), while $\vartheta = 20$ indicates a quality threshold of 20 (mDRACO behavior). $r =$ NR indicates that the pheromone association is not relative (standard ACO behavior), while $r =$ R indicates that it is relative (mDRACO behavior). The rightmost column contains the results for the full mDRACO algorithm

for $m = 17$ for the reasons described above; however, if we assume that it is at least 50,000s, mDRACO is at least 3000 times faster at $m = 17$.

Interestingly, arbitrary-length pheromone association on its own (the second column of Fig. 2) performs significantly worse than our implementation of standard ACO. To ensure that the good performance of mDRACO is not simply the result of the combination of the other two optimizations, we also conduct experiments using only relative path storage and a $\vartheta = 20$ quality threshold (the fifth column from the left in Fig. 2). This combination of optimizations is ~ 2.9 times slower than mDRACO for $m = 16$, ~ 7.5 times slower for $m = 17$, and $\sim$95 times slower for $m = 18$. We speculate that arbitrary-length pheromone association hurts ACO performance on its own but significantly improves run times when coupled with the other two optimizations because it makes the process of performing map lookups and insertions much more expensive, as every subsequence of the path generated by an ant is inserted to the map when a path is constructed, and every suffix must be checked when computing τ. This overhead overwhelms any benefits that can be gained from the additional information carried by pheromone when much effort is expended on the fruitless paths pruned early by a well-chosen ϑ.

4 Conclusion

We present a novel implementation of the ACO metaheuristic, mDRACO, that finds solutions to Costas-array problem for sizes $m \leq 19$ while maintaining high efficiency ($E > 0.5$) even at large worker counts ($N = 250$) and reduces median time-to-first solution by several orders of magnitude for relatively large problem sizes. We have applied mDRACO

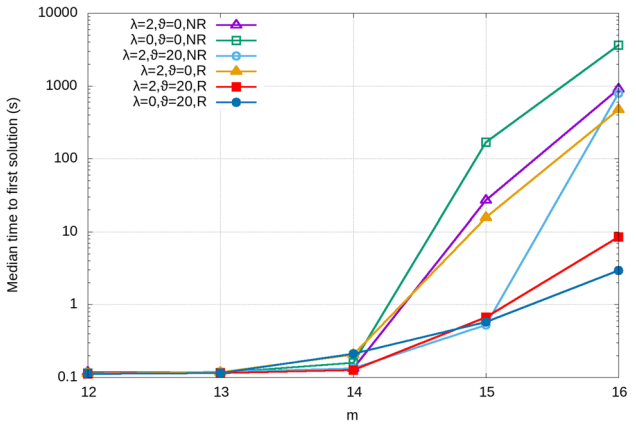

Fig. 3 Median time-to-first solution for our implementation of mDRACO over $12 \leq m \leq 16$, as in Fig. 2. The y-axis is logarithmic

to the unsolved $m = 32$ size of the Costas-array problem—at the time of writing, the ant system has used over 10,000 days of CPU time and discovered a permutation of the integers from 1 to 32 with 17 Costas-array-property violations (shown in "Appendix").

The program as written is generally applicable to any finite-domain CSP. We expect that the optimizations described in Sect. 2.4 generalize to some other CSPs and constraint-optimization problems. Specifically, we expect that arbitrary-length pheromone association (Sect. 2.4.1) improves the performance of ACO algorithms on combinatorial problems that can be solved with the typical TSP pheromone association scheme, such as the TSP itself, graph-coloring problems, and the N-Queens Problem. Relative path storage (Sect. 2.4.2) likely generalizes to problems in which constraints or costs relate to relative positions of domain values, such as the N-Queens Problem. Since the branch-and-bound quality threshold ϑ (Sect. 2.4.3) can be used to reduce wasted resources for any problem, we expect that it generalizes to all ACO solvers.

Further investigation is necessary to determine exactly which classes of combinatorial problems benefit from the three optimizations introduced by mDRACO, and to what extent. We also have not yet studied how these optimizations behave when used simultaneously with other known ACO variants, such as when they are coupled with local search heuristics, pheromone diffusion (Deng et al. 2019), or negative pheromone feedback (Ye et al. 2017). We have observed that the three optimizations introduced in this paper interact with each other in unexpected ways, so studying their interaction with other modifications may be an especially interesting direction of future research.

The C++ source for our mDRACO program is available at https://github.com/dvulakh/mDRACO.

Funding Open Access funding provided by the MIT Libraries. This study was funded by the Kentucky Young Researchers Program.

Data availability statement The software analyzed in this paper is available at https://github.com/dvulakh/mDRACO. The datasets generated and analyzed during the current study are available from the corresponding author on reasonable request.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors

Appendix: Arrays discovered

After over 10,000 CPU-days on a Knight's Landing computer, our mDRACO program has found these approximate solutions to the Costas-array problem with $m = 32$.

19 violations

8192520171013141626232283122464311130211529231185972227

18 violations

14122530131617262335207191093121824328274612921282215 11

9152028178222912253230161811213731191423127244261325106

14162342691019181117323115272112312824813230297255206 22

17 violations

27312318143152262964322522810247161751211202813021919 13

References

Arbelaez A, Codognet P (2014) A GPU implementation of parallel constraint-based local search. In: 2014 22nd Euromicro international conference on parallel, distributed, and network-based processing. IEEE, pp 648–655

Bouamama S, Blum C, Fages JG (2019) An algorithm based on ant colony optimization for the minimum connected dominating set problem. Appl Soft Comput 80:672–686

Bulatov AA, Krokhin AA, Jeavons P (2000) Constraint satisfaction problems and finite algebras. In: International colloquium on

automata, languages, and programming. Springer, Berlin, pp 272–282
Caniou Y, Codognet P, Richoux F, Diaz D, Abreu S (2015) Large-scale parallelism for constraint-based local search: the Costas array case study. Constraints 20(1):30–56
Cormen TH, Leiserson CE, Rivest RL, Stein C (2009) Introduction to algorithms. MIT press, Cambridge
Costas JP (1984) A study of a class of detection waveforms having nearly ideal range-Doppler ambiguity properties. Proc IEEE 72(8):996–1009
Deng W, Xu J, Zhao H (2019) An improved ant colony optimization algorithm based on hybrid strategies for scheduling problem. IEEE Access 7:20281–20292
Dorigo M, Di Caro G (1999) Ant colony optimization: a new meta-heuristic. In: Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406), vol 2. IEEE, pp 1470–1477
Dorigo M, Maniezzo V, Colorni A (1996) Ant system: optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybern Part B (Cybern) 26(1):29–41
Drakakis K (2011) Open problems in Costas arrays. arXiv preprint arXiv:1102.5727
Drakakis K, Iorio F, Rickard S, Walsh J (2011) Results of the enumeration of Costas arrays of order 29. Adv Math Commun 5(3):547–553
Golomb SW, Taylor H (1984) Constructions and properties of Costas arrays. Proc IEEE 72(9):1143–1163
Guan B, Zhao Y, Li Y (2019) An ant colony optimization based on information entropy for constraint satisfaction problems. Entropy 21(8):766
Guan B, Zhao Y, Li Y (2021) An improved ant colony optimization with an automatic updating mechanism for constraint satisfaction problems. Expert Syst Appl 164:114021
Liu Y, Chen Y, Jiang T (2018) On sequence planning for selective maintenance of multi-state systems under stochastic maintenance durations. Eur J Oper Res 268(1):113–127
Ning J, Zhang Q, Zhang C, Zhang B (2018) A best-path-updating information-guided ant colony optimization algorithm. Inf Sci 433:142–162
Roli A, Blum C, Dorigo M (2001) ACO for maximal constraint satisfaction problems. Proc MIC 1:187–191
Stützle T, Hoos HH (2000) MAX-MIN ant system. Futur Gener Comput Syst 16(8):889–914
Taylor K, Rickard S, Drakakis K (2011) Costas arrays: survey, standardization, and Matlab toolbox. ACM Trans Math Softw (TOMS) 37(4):1–31
Thiruvady D, Ernst AT, Singh G (2016) Parallel ant colony optimization for resource constrained job scheduling. Ann Oper Res 242:355–372
Truchet C, Richoux F, Codognet P (2013) Prediction of parallel speed-ups for Las Vegas algorithms. In: 2013 42nd International conference on parallel processing. IEEE, pp 160–169
Yan F (2018) Autonomous vehicle routing problem solution based on artificial potential field with parallel ant colony optimization (ACO) algorithm. Pattern Recogn Lett 116:195–199
Ye K, Zhang C, Ning J, Liu X (2017) Ant-colony algorithm with a strengthened negative-feedback mechanism for constraint-satisfaction problems. Inf Sci 406:29–41
Zhang H, Zhang Q, Ma L, Zhang Z, Liu Y (2019) A hybrid ant colony optimization algorithm for a multi-objective vehicle routing problem with flexible time windows. Inf Sci 490:166–190