Utility of Genetic Algorithms for Solving Large-Scale Construction Time-Cost Trade-Off Problems

Duzgun Agdas, Ph.D.[1]; David J. Warne[2]; Jorge Osio-Norgaard[3]; and Forrest J. Masters, Ph.D.[4]

Abstract: The time-cost trade-off (TCT) problem has long been a popular optimization question for construction engineering and management researchers. The problem manifests itself as the optimization of total costs of construction projects that consist of indirect project costs and individual activity costs. The trade-off occurs as project duration and, as a result, indirect project costs decrease with reduced individual activity duration. This reduction in individual activity duration is achieved by increasing resource allocation to individual activities, which increases their costs to completion. Historically, metaheuristic solutions have been applied to small-scale problems due to computational complexities and requirements of larger networks. Findings in this article demonstrate that the metaheuristic approach is highly effective for solving large-scale construction TCT problems. A custom genetic algorithm (GA) is developed and used to solve large benchmark networks of up to 630 variables with high levels of accuracy (<3% deviation) consistently using computational power of a personal computer in less than 10 min. The same method can also be used to solve larger networks of up to 6,300 variables with reasonable accuracy (~7% deviation) at the expense of longer processing times. A number of simple, yet effective, techniques that improve GA performance for TCT problems are demonstrated, the most effective of which is a novel problem encoding, based on weighted graphs, that enables the critical path problem to be partially solved for all candidate solutions a priori, thus significantly increasing fitness evaluation. Other improvements include parallel fitness evaluations, optimal algorithm parameters, and the addition of a stagnation criteria. This article also presents some guidelines of optimal algorithm parameter selection through a comprehensive parameter sweep and a computational demand profile analysis. Moreover, the methods proposed in this article are based on open source development projects that enable scalable solutions without significant development efforts. This information will be beneficial for other researchers in improving computational efficiency of their solution in addressing TCT problems. **DOI: 10.1061/(ASCE)CP.1943-5487.0000718.**

Introduction

The construction industry is characterized by nonrepetitive, complex projects, with generally unpredictable productivity figures. The level of complexity increases exponentially as the scope and size of projects increase. For various reasons, such as meeting deadlines to avoid liquidated damages and earn bonuses, crashing construction project schedules is a common practice in the construction industry. This is generally achieved through increased resource allocation to individual activities to reduce their duration. In most cases, both the increase in resources and reduction in duration are coupled in discrete pairs to simplify the problem (Sonmez and Bettemir 2012). The so-called time-cost trade-off (TCT) problem manifests itself in finding the optimum balance of total project costs. That is the balance of increased activity costs and reduced indirect costs because of reduced project duration as a result of reduced activity duration (Chassiakos and Sakellaropoulos 2005). In its most basic form, this problem can be represented as

$$C_T = C_A + C_I + D - I \tag{1}$$

where C_T = total project cost; C_A = total activity cost; C_I = indirect project costs; D = disincentives; and I = incentives. In reality, disincentives are more likely to be in a liquidated damage format that is applied when the project duration exceeds a predetermined threshold and incentives are bonuses rewarded for days saved from a specified threshold. The complication in minimizing the total project cost, C_T, arises due to the large number of time-cost pairs for individual activities, and the resulting network computations because of differing activity duration values. It is typical for large construction projects to consist of hundreds of activities. Considering the potential number of discrete time-cost pairs for these activities, combined with the logical sequencing of activities, the computational demands can be extreme.

Time-Cost Trade-Off Problem

Heuristic and metaheuristic methods have been the main tools in addressing this problem due to the previously mentioned computational requirements, although exact methods have also found some limited interest (Kandil et al. 2010; Sonmez and Bettemir 2012; Chassiakos and Sakellaropoulos 2005; Boussad et al. 2013). Of the metaheuristics methods used in construction and in general, genetic algorithms (GAs) have been undoubtedly the most prominent method used for a multitude of problems, including the TCT problem (Boussad et al. 2013). Inspired by the Darwinian theories on biology and evolution, GAs use genetic operations of crossover, mutation, and selection to find near-optimal solutions to otherwise complicated problems using an initial set of candidate solutions

[1]Senior Lecturer, School of Civil Engineering and Built Environment, Queensland Univ. of Technology, 2 George St., Brisbane, QLD 4001, Australia (corresponding author). E-mail: duzgun.agdas@qut.edu.au

[2]eResearch Analyst, High Performance Computing and Advanced Research Computing support, Queensland Univ. of Technology, 2 George St., Brisbane, QLD 4001, Australia.

[3]Graduate Student, Dept. of Civil, Environmental and Architectural Engineering, Univ. of Colorado Boulder, Boulder, CO 80309.

[4]Associate Dean, Herbert Wertheim College of Engineering, Univ. of Florida, Gainesville, FL 32611.

Note. This manuscript was submitted on August 14, 2016; approved on May 31, 2017; published online on October 31, 2017. Discussion period open until March 31, 2018; separate discussions must be submitted for individual papers. This paper is part of the ***Journal of Computing in Civil Engineering***, © ASCE, ISSN 0887-3801.

Table 1. Sample of Earlier Research Findings on TCT Problem Solution Benchmarks

Research	Network	Cost-time pairs	Method	Runtime (h)	Accuracy (%)	Parallelism
Feng et al. (1997)	18	5	GA	N/A	N/A	No
Hegazy (1999)	18	5	GA	0.11	N/A	No
Chassiakos and Sakellaropolos (2005)	29	3	LP/IP	<0.01	0.30	No
Kandil and El-Reyes (2006)	720	N/A	GA	6.70	N/A	Yes
Long and Ohsato (2009)	18	5	GA	N/A	N/A	No
Menesi et al. (2013)	2,000	5	Constrained programming	2.00	6.39	No
Sonmez and Bettemir (2012)	630	5	Hybrid GA	1.22	2.41	No
Ghoddousi et al. (2013)	37	2	GA	N/A	N/A	No
Monghasemi et al. (2015)	18	5	GA	N/A	N/A	No
Tran et al. (2016)	15	3	Hybrid GA	N/A	N/A	No

Note: LP/IP = linear programming/integer programming.

(Michalewicz 1999). Possible solutions are represented as a genome upon which genetic operations such as mutation and crossover can be performed, thus simulating the natural selection process.

Table 1 represents the state of the practice in regards to the TCT problem in the construction industry. It shows the typical size of the problems solved, the methods by which they are solved, performance of the methods used, and whether parallel computing was used to reduce the processing time. It is clear that the majority of the articles addressed smaller networks, and detailed performance benchmarks on solution accuracy and runtime performance also appear to be lacking.

In a broader context, the TCT problem can be considered a subset of the more general resource-constrained scheduling (RCS) problem, a heavily studied problem in operations research (Hartmann and Briskorn 2010; Vanhoucke and Debels 2007). In construction engineering and management literature, TCT and RCS problems appeared to have been treated as different topics (Kim and Ellis 2008; Zhang 2011); however, it is safe to say both belong under the same class of scheduling benchmark problems. The algorithms for RCS in areas of computer science, operation research, and operations management can be sophisticated (Debels and Vanhoucke 2007; Gaglioppa et al. 2008; He et al. 2017). Even in these disciplines, algorithm performance is often only reported for relatively small networks of less than 300 nodes. Therefore, demonstration of performance of large-scale, realistic, TCT problems is one important contribution of this work for construction engineering and management research and beyond.

Issues in Solving the TCT Problem Using GA

Computational complexity becomes relevant and problematic with the increased network size because cost computations are simple formulas. This can explain why the majority of the earlier studies—and this trend appears to continue in the more recent studies—on this subject has revolved around smaller networks (Feng et al. 1997; Hegazy 1999; Chassiakos and Sakellaropoulos 2005; Long and Ohsato 2009; Ghoddousi et al. 2013; Monghasemi et al. 2015; Tran et al. 2016). Feng et al. (1997) and Chassiakos and Sakellaropolos (2005) addressed networks of 18 and 29 variables, respectively. These two articles represent some of the more popular networks that have later been used in the literature as key benchmark problems. With the advancements in computational efficiency and better understanding of these problems, larger networks have also been analyzed (Kandil and El-Rayes 2006; Kandil et al. 2010; Sonmez and Bettemir 2012). Perhaps the most interesting article on the scale of problems addressed in the literature has been the publication by Menesi et al. (2013), in which the authors criticize the earlier literature because of the size of the networks analyzed and urge other researchers to focus on methods that can provide fast, accurate solutions to large-scale TCT problems. Following this criticism of Menesi et al. (2013), this article is an attempt to improve the state of GA optimization practice by taking a more holistic approach to performance analysis that considers accuracy, convergence time, computational efficiency, and ease of development. Thus, findings presented are highly relevant for real-life-scale applications; in particular, the approach to the parameter selection step should serve as a useful guide for practical use of GA for construction TCT problems.

Research Motivation and Contribution

It is clear from the existing literature that there is a lack of GA exemplars that could demonstrate the capacity to solve realistic large-scale construction TCT problems efficiently to make them accessible for real-life applications. The term *efficiency* is used here to indicate solutions with reasonable computational requirements that can be met by a personal computer and processing time (~minutes). The main motivation behind this research is to address this gap, and following are the main research questions:

- How feasible, in terms of computational demand and accuracy, is the application of metaheuristics to realistic TCT problems?
- What features of the TCT problem can be exploited to improve the efficiency of metaheuristics?
- What is the software development effort required to achieve such solutions?

Motivated by these research questions, this article provides the following contributions to the construction industry and research community:

- The presentation of a holistic analysis on the utility of GA to solve real-world TCT problems. This analysis considers the trade-off of solution optimality in terms of time to solution, the complexity of implementation, and computer hardware and software requirements.
- The presentation of a number of well-studied, but highly effective, methods for improving GA performance of TCT problems. These include optimization of fitness evaluations using weighted graphs, a population stagnation criterion, and implementing parallel computing.
- The provision of guidelines for GA parameter selection for TCT problems based on parameter performance assessment using approximate Bayesian techniques.
- The demonstration of the efficacy of freely available, standard off-the-shelf GA implementations in solving realistic TCT problems typical of real construction projects.

Organization of the Article

The following section of the article describes the research methodology; the benchmark models used, solution method, and implementation approach are discussed in detail. An improved GA

design to solve TCT problems is then introduced. This section includes discussions on how to encode the problem, evaluate fitness, and use a stopping criteria to maximize the computational efforts. Also described in this section is a novel approach to the fitness assessment step of GA development. The results section includes discussions on how to solve the benchmark problems where the model performance is assessed in terms of accuracy, processing time, and processing demand. The article concludes with the summary and long-term implications of the findings.

Research Methodology

Defining the Problem: Benchmark Studies

The majority of the literature reviewed use the fundamental problems defined by Feng et al. (1997) and Chassiakos and Sakellaropoulos (2005). This article is structured around the 18- and 63-variable problems defined by Feng et al. (1997) and Sonmez and Bettemir (2012) in devising larger networks. This was achieved by repeating these networks in series to achieve larger networks, and because the solution to smaller networks are known, so are the solutions to the larger networks. This property provides the benchmark for accuracy comparisons. Although the networks created in this fashion may not capture the actual complexity of networks of comparable size, this method has been used and accepted consistently in earlier literature (Aminbakhsh and Sonmez 2016).

Algorithm development efforts in this article were structured around the 63-variable networks defined by Sonmez and Bettemir (2012). These two topologically identical networks (63a and 63b) have a minor difference in their respective indirect cost figures. Additionally, results for analyses of larger networks of 630, 1,800, 3,150, and 6,500 variables are also provided. Sonmez and Bettemir (2012) provide an interesting perspective on the increased computational complexity as the number of time-cost pairs associated with these networks—the computational complexity of these networks are likely to increase exponentially with increased activity numbers. For the 18-variable problem there are 5.9×10^9 discrete time-cost alternatives, while the same for the 63-variable problem are 1.4×10^{42}. For 180- and 630-variable networks, these values are 5.2×10^{97} and 2.9×10^{421}, respectively. These values only indicate the mode selection for activities; the topology sorting for larger networks (i.e., critical path computations) will also be more complicated.

Solution Method: Genetic Algorithms

Genetic algorithms have long been the most common and widely used metaheuristics in construction engineering and management research. However, examples of more sophisticated metaheuristics solutions such as particle swarm optimization (Zhang et al. 2005), shuffled frog leaping (Fang and Wang 2012), ant colony optimization (Merkle et al. 2002), and memetic algorithms (Senouci and Eldin 2004) also exist (Zhang and Ng 2012). Despite this popularity, no clear benchmarks exist on their design and capacity as potential practical solutions to large-scale TCT problems outside a few studies (Aminbakhsh and Sonmez 2016; Menesi et al. 2013; Kandil et al. 2010). Thus, this article provides analysis of GAs performance across different benchmark problems as a starting point to test their performance.

Solution Platform

An important goal of this article was to assess the GA performance without devising purpose-specific algorithms in an attempt to minimize the development efforts and ensure extensibility of the proposed solutions to different problems. The application platform chosen for this study was to use a generic GA solution through the Distributed Evolutionary Algorithms in Python (DEAP) framework (Fortin et al. 2012). This open source framework is built on the Python programming language—an open source, high-level programming language (*Python*). DEAP was chosen due to the ease of development and integration of parallel computing. The method chosen for parallelization was Scalable COncurrent Operations in Python (SCOOP) within the DEAP framework due to its seamless integration with DEAP and relatively low cost of development in implementation (Hold-Geoffroy et al. 2014).

Table 2. Preliminary GA Benchmarks with DEAP and SCOOP

Size	DEAP runtime (h)	DEAP accuracy (%)	DEAP + SCOOP runtime (h)	DEAP + SCOOP accuracy (%)
630a	17.45	0.43	4.05	0.80
630b	18.98	0.83	4.61	1.20

Preliminary Implementation and Observations

To assess the baseline performance of DEAP as a general implementation method and SCOOP as the parallel computing medium, preliminary optimization instances were run. No modifications were made to the DEAP code to implement the problem to assess its viability for scalability and reduce development efforts. A reasonable assumption was made about the upper limits of the population size (2,000) and the number of generations (2,000) (Kandil and El-Rayes 2006). The default GA parameters were mutation rate of 0.1 and polynomial bounded method (Gaussian, shuffle indexes, flip bit, and uniform methods were also available), crossover rate of 0.5, and uniform partially matched method (one- and two-point, uniform, partially matched, ordered, and blend methods were also available), and Nondominated Sorting Genetic Algorithm II (NSGA-II) selection method [tournament, roulette, Strength Pareto Evolutionary Algorithm II (SPEA-II), random, best, worst, and tournament Dominance and Crowding Distance (DCD) methods were also available] (Fortin et al. 2012). A snapshot of the overall performance of the results of these runs is given in Table 2. The reported baseline performance values are for the 630-variable problem because this was the most recent benchmark problem, and more detailed analyses for different networks are provided subsequently.

Remarkably, the results were extremely accurate, exceeding those reported by Sonmez and Bettemir (2012); however, the computational time, on average, was approximately 18 h for the five routines run for each of these test instances. Due to the expected nonlinear growth in processing time of the larger network problems, they were not attempted using this method. Having arrived at an accurate DEAP implementation of the benchmark problem by Sonmez and Bettemir (2012), this research was extended to exploit parallel computing using SCOOP. This resulted in more than 4× speedup over the serial code using eight CPU cores; however, the overall runtime (4 h) indicated that solving realistic TCT problems would still be infeasible.

Improved Genetic Algorithm Design

The preliminary analysis indicated that the DEAP package can provide accurate solutions, although the processing times were still significant. Parallel computing with the SCOOP package also

reduced the computational time significantly when the computations were distributed over multiple CPUs. Considering these observations, the following sections of this article show further improvements made to the GA design—including introducing a stopping criterion, details of the graph-based fitness evaluation method, and selection of optimal GA parameters—to minimize computational time without sacrificing accuracy.

Problem Encoding

Arguably, the most important element in effective use of a GA is the encoding of a problem solution as a genetic code. In terms of biological analogy, the problem solution can be considered as the organism or the phenotype and the genetic code is the genotype. The method of reading a genome to produce a problem solution is the embryology. The choice of encoding along with the associated embryology, crossover, mutation, and selection operations can have a significant effect on the effectiveness of the method (Affenzeller et al. 2009). A simple encoding approach was determined to be effective in modeling the TCT problem, and the genotype of a individual is defined as a sequence of construction modes.

First, consider the time-cost optimization problem with n activities $A = \{a_1, a_2, \ldots, a_n\}$, where a_i is an integer that represents the number of modes that can be assigned to the ith activity. Each mode assignment, $m_{i,j}$, is a cost-duration pair, $m_{i,j} = (c_{i,j}, d_{i,j})$, where $c_{i,j}$ and $d_{i,j}$ are respectively the cost and duration of the ith activity using the jth mode. The genotype for a potential solution set of mode assignments is given by the list $G = \{g_1, g_2, \ldots, g_n\}$, where $1 \le g_i \le a_i$ for $i = 1, 2, \ldots, n$. Here each gene, g_i, encodes the mode number for activity i.

The phenotype is constructed by assigning activity i to the g_ith possible mode. This yields an embryology function $E(G)$ defined by

$$E(G) = \{(1, g_1), (2, g_2), \ldots, (n, g_n)\} \tag{2}$$

A fitness function can then be determined in terms of the phenotype and genotype.

Fitness Evaluation

The fitness of an individual phenotype $p = E(G)$ is the total cost of the construction process given the activity mode assignments. In this context, an assignment is defined as an activity-mode pair. For example, if activity i will implement mode g_i, then the mode assignment is $p_i = (i, g_i)$. This total cost, C_T, consists of the sum of the activity costs, C_A, and the indirect costs, C_I, of the total project duration. That is

$$C_T = C_A + C_I \tag{3}$$

This is simply Eq. (1) without the incentive and disincentive terms—these terms can be simply added to the formula if the problem characteristics require.

Given individual p, C_A, and C_I can be calculated as

$$C_A = \sum_{i=1}^{n} c_{p_i} \tag{4}$$

$$C_I = r \sum_{j \in L} d_{p_j} \tag{5}$$

where r = indirect cost rate (\$/time); and L = set of activities that contribute to the critical path of the project.

By substitution of Eqs. (4) and (5) into Eq. (3), a fitness function can be defined

$$F(p) = \sum_{i=1}^{n} c_{p_i} + r \sum_{j \in L} d_{p_j} \tag{6}$$

This can, in turn, be expressed in terms of the genotype using the embryology [Eq. (2)]

$$F^*(G) = \sum_{i=1}^{n} c_{i,g_i} + r \sum_{j \in L} d_{j,g_j} \tag{7}$$

Computational complications arise from computing the latter part of the formula caused by the necessity to compute the longest path (i.e., critical path) of the network analyzed. A significant contribution of this article to general construction engineering and management literature is the computation of the critical path using a robust and efficient method from graph theory. This method removes the need for repeated topology sorting with different time-cost mode allocation and improves overall computation time substantially. Used in various other disciplines such as operations management (e.g., minimum spanning tree), graph theory is the mathematics of objects with interconnected relationships (Floudas and Pardalos 2008). Although graph theory has been applied to general scheduling problems (Dharwadker and Pirzada 2011), no such study in construction engineering and management literature exists to the best of the authors' knowledge. The main advantage of using a graph-based solution to the classical TCT problem is that the requirement for topological sorting of the network needs only be completed once, rather than the more traditional approach of iteratively constructing the critical path for each one of the GA solutions. Because the time-cost pairs are assigned to activities in no particular order in search of the lowest project costs, all corresponding topologies need to be sorted (i.e., critical path) at each step of genetic operations. It is not clear whether this iterative approach was replicated in earlier literature, but this method was used in the earlier optimization analyses conducted for this article. This also explains the significant computational time associated with these optimization runs.

Stopping Criteria

A challenge in using GA is to determine when to terminate (Affenzeller et al. 2009; Kim 2013) the genetic operations. This is a trade-off between the quality of the solution and the overall execution time. With everything kept equal, more generations will always lead to better solutions unless the correct answer is found during computations; however, the rate at which this improvement occurs is greatly dependent on the genetic diversity in the population of solutions. When the diversity of the solutions gets low, the populations stagnate and the solutions are essentially stabilized. At this point there is little to be gained by continued genetic operations, and it is sensible to terminate the evolution to optimize the required computation time. To achieve this, a measure of diversity that compares the relative distance between fitness of the best individual to the average fitness of the whole population was devised

$$S(P) = 1 - |P| \frac{\min_{G \in P} F^*(G)}{\sum_{G \in P} F^*(G)} \tag{8}$$

where P = gene pool, that is, the set of all genomes in the current population. This stagnation criterion is used to determine the trade-off between computational effort and solution accuracy. In this article, the stagnation threshold of $T = 5 \times 10^{-5}$ and the stopping criterion of $S(P) < T$ were used.

Critical Path Computations

The total cost function [Eq. (3)] and derived fitness function [Eq. (7)] consist of two distinct components: the activity costs and the indirect cost. Of these, the indirect cost is more computationally expensive due to the requirement that the critical path (i.e., duration) of the project be calculated to determine the total project duration. The reviewed construction engineering and management literature has not explicitly stated the method used in computing the project duration, but it appears this is computed as the longest path within a network (Sonmez and Bettemir 2012). This can simply be calculated by iteratively computing the completion time of activities using the precedence logic. In this article, the preliminary computations were run using this method. Once an individual activity time-cost pair is selected from available modes for an activity, the time is registered as the accepted activity duration and fed into the critical path computations. This is identical to the forward pass computations of the traditional critical path method (CPM). Although effective in providing a simple solution to project duration computations, this approach proves to be inefficient due to the excessive time required to compute the results, as in each iteration the network topology needs to be recalculated because of the activity mode selection.

Longest Path Problem

The critical path problem is a specific case of a longest path problem. This widely studied fundamental problem of finding the longest path within a graph is essentially the very description to finding the longest path within a given schedule (Floudas and Pardalos 2008). Studying the effective solution algorithms devised from the longest path problem, a computationally efficient solution for the TCT problems under consideration was developed.

Graph Method

A graph is defined as a set of vertices, V, and a set of edges, E. For an undirected graph, vertices $u, v \in V$ are said to be connected if there is an edge $e \in E$ such that $e = (u, v)$. The neighborhood of vertex v by $n(v)$, that is, $n(v)$, is the set of all vertices in E that are connected to v. A weighted graph also includes a function $\omega(e)$ that assigns edges to a measure of distance between its end points. Importantly, a mode's duration through the weighting function $\omega(e)$ was explicitly encoded, which enables efficient preprocessing via a topological sort.

A path is defined as a connected sequence of vertices $P_{v_0,v_n} = (v_0, v_1, \ldots, v_n)$, that is, for all $i \in [0, n-1]$ there is an edge $(v_i, v_{i+1}) \in E$. The length of a path, denoted by $D(P_{v_0,v_n})$, is the sum of the edge weights connecting the vertices in the path, that is, $D(P_{v_0,v_n}) = \sum_{i=0}^{n-1} \omega[(v_i, v_{i+1})]$.

The longest path problem can be stated as follows: Given a weighted graph, $G = (V, E)$, with weighting function $\omega(e)$ and two vertices $s, t \in V$, the task is to find a path $L_{s,t}$ that satisfies

$$L_{s,t} = \underset{P_{s,t} \in \Phi_{s,t}}{\operatorname{argmax}} D(P_{s,t}) \tag{9}$$

where $\Phi_{s,t}$ = set of all possible paths between s and t. For a general graph G, the longest path problem has been shown to be non-deterministic polynomial-time (NP)-hard—similar to the TCT problem as noted by Sonmez and Bettemir (2012).

Solution for Directed Acyclic Graphs

There are two special types of graphs: directed and acyclic graphs. A directed graph defines connectivity as unidirectional, that is, if $(u, v) \in E$ then u is connected to v but v is not connected to u. A graph is acyclic if there do not exist any paths with $v_0 = v_n$; such a path is called a cycle. Although the longest path problem is NP-hard for a general weighted graph, an efficient algorithm exists for graphs that are directed and acyclic with a nonnegative weight function $\omega(e) \geq 0$ for all $e \in E$ (Kahn 1962; Knuth 1968). The algorithm consists of two steps: (1) sort the vertices topologically, and (2) visit vertices in order and compute longest path as the longest path from each incoming edge.

A vertex list of a directed acyclic graph (DAG) is topologically sorted if for every directed edge $e = (v_i, v_j)$, then $i < j$. That is, all vertices from all possible paths leading to the jth vertex are visited before the jth vertex itself. A topological sort can be computed using the algorithm shown in Fig. 1. A topological sort is only valid for directed acyclic graphs. Once the topological sort has been applied, the longest path between all nodes can be computed using the algorithm shown in Fig. 2.

This longest path algorithm has a computational complexity of $O(|V| + |E|)$. Because the activity dependency graph for a TCT problem is directed, acyclic, and activity durations are always nonnegative, the algorithm shown in Fig. 2 is directly applicable.

Thus, the fitness function $F^*(g)$ [Eq. (7)] can be computed efficiently in $O(|V| + |E|)$ operations (Cormen et al. 2001). Fig. 3 demonstrates the relative proportion of CPU time spent in fitness evaluations using the graph-based scheme versus the original, iterative approach. This figure shows computational demands for

```
input  : Directed Acyclic Graph D = (V,E)
output: Directed Acyclic Graph D' = (V',E), such that V' is topologically sorted
V' = ∅;
Q = {v : v ∈ V,(∀u ∈ V,(u,v) ∉ E)};
while |Q| ≠ 0 do
    v ∈ Q;
    Q ← Q − v;
    if v ∉ V' then
        V' ← V' ∪ {v};
    end
    for u such that (v,u) ∈ E do
        if (x,u) ∈ E implies x ∈ V' then
            V' ← V' ∪ u;
        end
    end
end
```

Fig. 1. Topological sort algorithm

input : Directed Acyclic Graph $D = (V, E)$ with weight function $\omega : E \to \mathbb{N}$
output: Array of predecessors V_p in the longest path for each vertex $v \in V$
for $v \in V$ **do**
  $V_p[v] \leftarrow \text{argmax}_{u \in v(v)}\, d[u] + \omega(u, v)$;
  $d[v] \leftarrow \max_{u \in vv}\, d[u] + \omega(u, v)$;
end

Fig. 2. Longest path algorithm for a topological sorted, nonnegatively weighted DAG

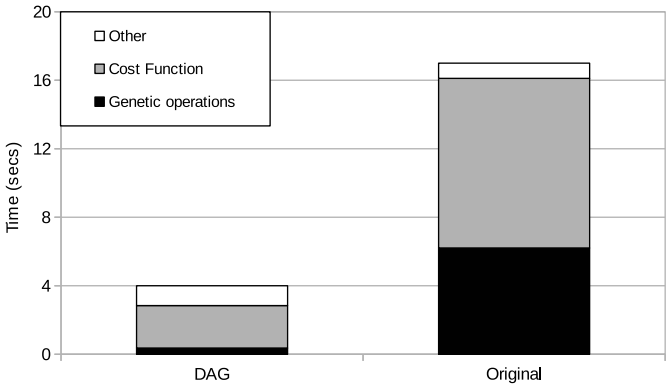

Fig. 3. Computational demand profile for GA implementation

different parts of the overall computations and the advantage of the DAG method over the traditional approach to the 63a problem.

DAG Implementation Using DEAP and SCOOP

A parallel genetic algorithm for solving the TCT using *Python* was implemented, in which the computationally demanding critical path durations were calculated using the graph theory. The DEAP library was utilized for the genome definition population creation and genetic operations (Fortin et al. 2012). The fitness evaluation step, which is the most computational task, has been parallelized using the SCOOP library (Hold-Geoffroy et al. 2014).

At the beginning of each iteration, individuals were randomly paired together based on fitness based on one of several selection methods. Crossover is then performed with each of these pairs to produce exactly two offspring, upon which mutation is applied at random. Finally, all offspring and parents are pooled into one population, from which the best 50% are selected for the next generation. This cycle repeats until a fixed generation limit is reached or if the population stagnates [as computed by Eq. (8)]. An overview of this approach is given in Fig. 4.

The most efficient method of computing the longest path in a DAG consists of a topological sort followed by an incremental build of the longest path by choosing the longest incident path at each node. Because individual activity dependencies do not change with different activity duration-cost pair selection, the topological sort needs to be applied once. This dramatically increases the overall performance of the critical path computations. Larger problems were solved efficiently by removing repeated topological sorting. Because each fitness evaluation is completely independent, these can be executed in parallel using the *Python* SCOOP library (Hold-Geoffroy et al. 2014).

Parameter Selection

Performance (computation time and accuracy) of GAs is affected by the parameter selection, and to the best of authors' knowledge no comprehensive parameter sweep of GA parameters has been conducted on TCT problem solution criteria—examples of such studies exist for different scheduling problems (Alcaraz and Maroto 2001). To address this research gap, approximate Bayesian computation (ABC) techniques were used to analyze the algorithm parameter distribution for a given desired accuracy threshold. The vector of algorithm parameters, θ, was treated as a random variable with probability distribution $p(\theta)$, then the parameter distribution was computed, in which the solution under a given parameter set, S_θ, is the exact minimum solution, S_M using Bayes' theorem

$$p(\theta \text{ given } S_\theta = S_M) = \frac{p(S_\theta = S_M \text{ given } \theta)p(\theta)}{p(S_\theta = S_M)} \tag{10}$$

While Eq. (10) is not tractable, it can be estimated for a tolerable level of accuracy, ϵ, through GA simulations with randomized parameters. This method, given in Fig. 5, is known as the ABC rejection sampling in the computational statistics literature (Sunnaker et al. 2013).

This ABC rejection scheme was applied using the 63a and 63b TCT problems using a target accuracy of $\epsilon = 0.7\%$ with uniform initial parameter distributions. For the crossover and mutation rates, this was continuous uniform distributions in the interval [0,1], and the crossover, mutation, and selection functions were treated as uniform discrete random variables (i.e., each was selected with equal probability from the available options within DEAP documentation). These values were used for the limits for the initial parameter distribution for ABC analysis. After computing 300 accepted ABC samples, the optimum parameters in balancing accuracy and processing time using the mean of the ABC samples were determined. For the selection, crossover, and mutation functions, this mean was rounded and found the tournament selection function, the one-point crossover function, and the shuffle indexes mutation function to be optimal. The optimal mutation and crossover rates were found to be 0.107 ± 0.079 and 0.474 ± 0.082, respectively. The ABC analyses in this article exclude the number of generations run and the population size because they were discussed in previous parts of this article. A particularly interesting observation is the lack of performance from widely popular functions such as NSGA-II and SPEA-II. This can be explained in part because these algorithms are suitable for multiobjective optimization, and the TCT problems presented here are not and can be solved with simpler and faster-converging algorithms.

Results and Discussions

Performance

Solution Accuracy

The largest network with clear accuracy numbers has been provided by Sonmez and Bettemir (2012), using their developed hybrid GA, as 2.41 and 2.47% for 630a and 630b problems. The algorithms used in this research produced significantly better

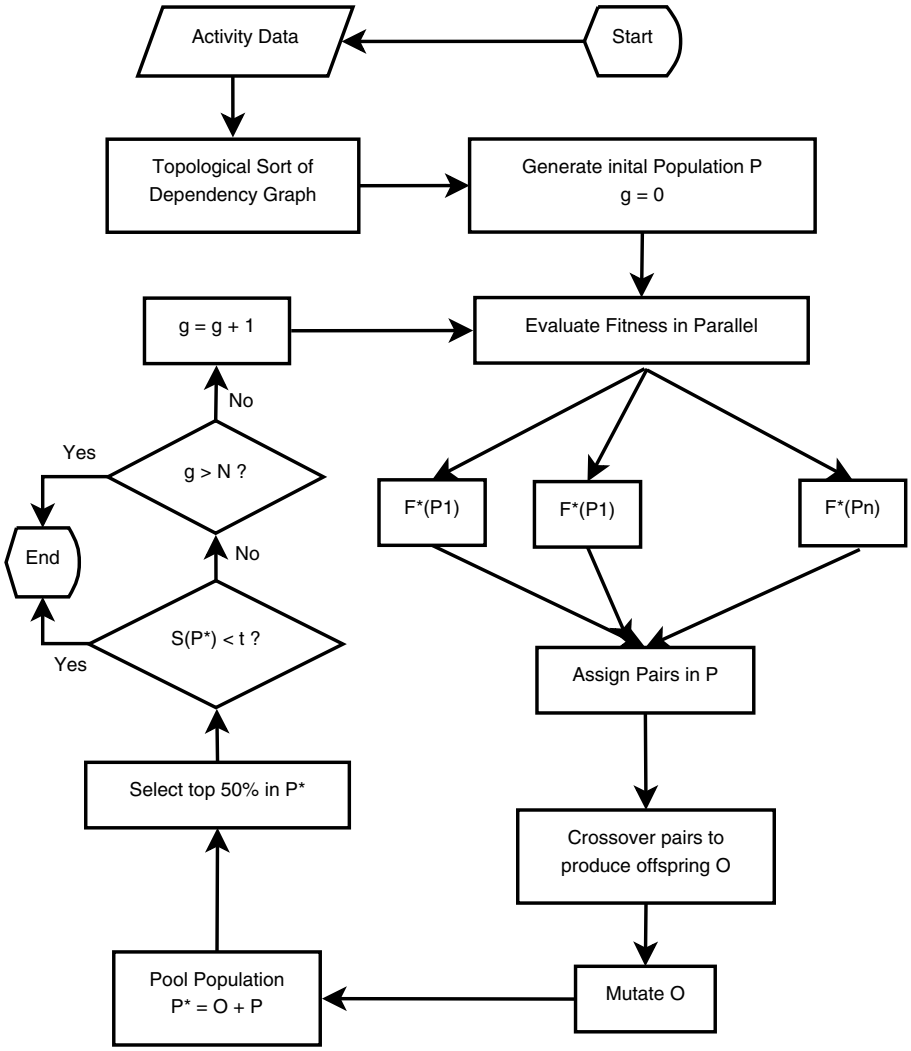

Fig. 4. Flowchart of GA implementation

input : Initial parameter distribution $p(\theta)$ and accuracy tolerance ϵ
output: Generates samples, $\theta^1, \theta^2, \ldots, \theta^N$, of the parameter distribution with accuracy ϵ
for $i = 1, 2, \ldots, N$ **do**
  repeat
    Generate sample of θ^* using $p(\theta)$;
    Compute TCT solution, C_T, using GA algorithm with parameters θ^*;
  until $C_T \leq \epsilon$;
  $\theta^i \leftarrow \theta^*$;
end

Fig. 5. ABC rejection sampling for optimal algorithm parameter selection

results for the same networks during parameter selection analyses described in previous sections of this article. The average accuracies were 0.79% (minimum = 0.43%, maximum = 1.57%) and 1.12% (minimum = 0.64%, maximum = 1.71%) for the 630a and 630b problems, respectively. The results were encouraging and clearly indicate the efficacy of the proposed optimization routine in achieving extremely accurate solutions to a large-scale TCT problem. However, for any solution of this problem to be practically viable, accuracy needs to be accompanied by convergence speed, as well as development effort—a factor mostly ignored in earlier literature. Because of these encouraging results and significantly faster convergence rates, larger networks of 1,800, 3,150, and 6,300 variables were solved to test the convergence limits of the algorithms.

No stagnation criteria were used in these experiments because the goal was to assess the limits of convergence, rather than practical solutions. The analyses were run five times for each of these instances and average values for both accuracy and runtime are reported in Table 3. The number of generations was set to 5,000 with a population size of 2,750 to test the accuracy limits of the proposed solution. These test instances were run with eight CPUs. The results of these larger networks are also encouraging with the exception of

Table 3. Processing Time and Solution Accuracy for Very Large Networks

Network	Runtime (h)	Accuracy (%)
1800a	5.82	7.05
1800b	5.84	14.72
3150a	9.15	6.50
3150b	9.41	4.73
6300a	16.42	7.66
6300b	16.76	6.96

an 1,800-variable network. There are no clear explanations as to why this variability of the accuracies was observed while processing times were comparable to the network sizes. Another interesting observation is that the growth in processing time appears to be near linear with increased network size. This is a clear indicator of the efficiency of the proposed method in addressing the computational complexity of the problems under assessment. The observed variable accuracy might have been caused by the inherent variability of the results consistency of metaheuristic methods for different problems or that the complexities of solution surfaces of the original 18-variable problem might be more complicated than those of the 63-variable problem, although the latter is the much larger network.

Development Effort

It is common in construction engineering and management research to use existing, open source optimization routines (Kandil and El-Rayes 2006) or purpose-specific software platforms (Menesi et al. 2013), or to develop customized optimization routines (Sonmez and Bettemir 2012) using different software packages and programming languages. In this article, DEAP, an open source development project aimed at increasing the viability of evolutionary algorithms written in *Python*, was used. In developing solutions for the TCT and parallelism, the main goal was to minimize the development efforts and rely as much as possible on off-the-shelf solutions. Similar to GA development, the SCOOP package was used for parallel computing, which was built into the DEAP package with minimal alterations.

Runtime Performance and Processing Demand

Kandil and El-Rayes (2006) cited a TCT problem of a 720-variable network that took 136.5 h to solve, which was reduced to 19.72 h using parallel computing and 6.70 h when using coarse-grained parallel modules. In a follow-up article, Kandil et al. (2010) reported a 90-h runtime without parallel computing that can possibly be attributed to faster processors. Kandil et al. (2010) also provide a processing time versus number of processors to evaluate their relationship. Following this approach, a comparison of reduced processing time with added computational resources through parallel computing is provided. Table 4 summarizes the runtime performance with added computational demand across multiple levels of accuracy. It can be seen that problems of 630 variables are solved in less than 10 min consistently using no more than eight CPUs. This is significant because these results indicate a more than 100× speed increase to the nongraph, nonparallel GA approach used previously in this article (Table 2).

In parallel computing, the limitations of parallelization must also be considered. The maximum speedup of a parallel algorithm is constrained by the ratio of strictly serial portions of the code. This is generally referred to as Amdahl's law (Amdahl 1967). Essentially, this is an example of the law of diminishing returns. As the parallel portion of the code is increased, the strictly serial portion becomes a bigger proportion of the total runtime. As the number of cores goes to infinity, the limit of the speedup is the original serial code runtime divided by the strictly serial portion of the parallel code—theoretically, this is the maximum possible parallelization. During the analyses, one to eight CPUs were considered. In a majority of these runs, no noticeable improvements in either time or accuracy were noted when more than six CPUs were used. The algorithms for this research effort were developed using a state-of-the-art high-performance computing (HPC) facility, which creates questions about how viable this method is for implementation on a personal computer. To test the viability of solving these problems on a personal computer, some of these test instances were also run on a personal computer. This PC had a six-core Intel (Santa Clara, California) Xeon CPU (W3670 clocked at 3.2 GHz) with

Table 4. Runtime and Accuracy versus Number of CPUs

Stopping criteria	Network	CPUs	Runtime (min)	Accuracy (%)
2,000 generations	63a	2	29.04 ± 0.31	0.28 ± 0.26
2,000 generations	63a	4	20.74 ± 0.65	0.34 ± 0.14
2,000 generations	63a	6	18.93 ± 0.38	0.33 ± 0.38
2,000 generations	63a	8	18.32 ± 0.17	0.48 ± 0.53
2,000 generations	63b	2	29.01 ± 0.73	0.75 ± 0.31
2,000 generations	63b	4	21.88 ± 2.78	0.96 ± 0.18
2,000 generations	63b	6	19.06 ± 0.32	0.89 ± 0.44
2,000 generations	63b	8	18.39 ± 0.13	1.20 ± 0.37
2,000 generations	630a	2	125.72 ± 5.38	1.47 ± 0.25
2,000 generations	630a	4	99.38 ± 11.80	1.62 ± 0.25
2,000 generations	630a	6	77.14 ± 2.24	1.78 ± 0.31
2,000 generations	630a	8	73.92 ± 0.07	1.93 ± 0.61
2,000 generations	630b	2	123.25 ± 2.08	1.84 ± 0.47
2,000 generations	630b	4	94.72 ± 9.78	2.20 ± 0.47
2,000 generations	630b	6	76.76 ± 1.86	2.30 ± 0.25
2,000 generations	630b	8	71.99 ± 1.79	2.47 ± 0.17
Stagnation	63a	2	2.15 ± 0.17	0.27 ± 0.18
Stagnation	63a	4	2.06 ± 0.28	0.39 ± 0.33
Stagnation	63a	6	1.91 ± 0.18	0.37 ± 0.25
Stagnation	63a	8	1.85 ± 0.13	0.26 ± 0.18
Stagnation	63b	2	2.00 ± 0.37	0.93 ± 0.64
Stagnation	63b	4	1.38 ± 0.42	1.84 ± 1.92
Stagnation	63b	6	1.73 ± 0.26	0.64 ± 0.32
Stagnation	63b	8	1.48 ± 0.23	1.24 ± 0.59
Stagnation	630a	2	10.08 ± 6.15	3.15 ± 3.44
Stagnation	630a	4	9.38 ± 4.13	2.25 ± 1.49
Stagnation	630a	6	5.11 ± 3.34	4.37 ± 3.11
Stagnation	630a	8	6.07 ± 2.27	2.76 ± 1.64
Stagnation	630b	2	8.93 ± 2.93	2.49 ± 0.83
Stagnation	630b	4	6.63 ± 1.38	2.63 ± 0.48
Stagnation	630b	6	4.73 ± 2.80	3.77 ± 2.60
Stagnation	630b	8	6.61 ± 0.77	2.29 ± 0.41
Literature accuracy	63a	2	1.65 ± 0.10	1.39 ± 0.25
Literature accuracy	63a	4	1.55 ± 0.06	1.33 ± 0.12
Literature accuracy	63a	6	1.54 ± 0.13	1.40 ± 0.17
Literature accuracy	63a	8	1.39 ± 0.08	1.50 ± 0.24
Literature accuracy	63b	2	1.68 ± 0.28	1.42 ± 0.15
Literature accuracy	63b	4	1.46 ± 0.12	1.65 ± 0.28
Literature accuracy	63b	6	1.50 ± 0.14	1.50 ± 0.24
Literature accuracy	63b	8	1.38 ± 0.19	1.50 ± 0.25
Literature accuracy	630a	2	10.52 ± 1.10	2.18 ± 0.15
Literature accuracy	630a	4	7.63 ± 0.64	2.18 ± 0.10
Literature accuracy	630a	6	7.10 ± 0.63	2.23 ± 0.06
Literature accuracy	630a	8	7.37 ± 1.26	2.33 ± 0.30
Literature accuracy	630b	2	10.36 ± 0.98	2.30 ± 0.13
Literature accuracy	630b	4	7.78 ± 1.13	2.32 ± 0.25
Literature accuracy	630b	6	7.09 ± 1.78	2.47 ± 0.33
Literature accuracy	630b	8	6.13 ± 0.58	2.43 ± 0.24

Note: Each of these test instances are run five times, and what is reported in the runtime and accuracy columns is the average of these runs and half of the range (maximum-minimum) for each. This is done to indicate the approximate distances from the mean for each one of these runs.

16 GB of RAM and the operating system was the *Red Hat Enterprise Linux*. This is a relatively old (~2010) system that performed on par with the HPC runs for larger problems, indicating that common PCs can be used in applications based on the methodology proposed here.

Strengths and Weaknesses of the Proposed Solution

The DAG model implemented in DEAP and SCOOP can be a viable solution to large-scale discrete TCT problems that are common in the construction industry. The implemented model was developed with minimum coding and software development using open source tools, and processing demands can be met by a personal computer while keeping the computational time manageable. The solution developed in this article compares favorably in terms of accuracy and processing time with the earlier studies. Development effort comparisons are more challenging because specifics of earlier studies are not detailed to provide such a comparison. It can be deduced from the earlier articles that researchers have developed problem-specific GA solutions (Sonmez and Bettemir 2012) and built on existing algorithms (Kandil et al. 2010); however, there are no discussions on the development efforts needed. The parameter sweep analyses conducted indicate that the model performance will be heavily dependent on problem formulation and GA parameter selection. This is, perhaps, also the biggest weakness of the algorithms developed in this article because their performance is not consistent across different problems and users will have to make decisions on what parameters should be selected. The proposed solution here is a steep improvement from traditional GA solutions to the TCT problem; however, there are other solutions that can solve the TCT problem (Aminbakhsh and Sonmez 2016). The solution set presented in this article is not mutually exclusive to other efforts because the DAG method can be implemented in the longest path computation step of different algorithms.

Conclusion

This article presents an innovative approach to solving large-scale construction time-cost trade-off problems using GAs that can solve real-life-scale TCT problems with exceptional accuracy, while not creating excessive computational demand or processing time. This is achieved by using a well-studied, yet innovative, method of graph theory in solving the critical path problem. The computational bottleneck in solving large-scale TCT problems is the fitness assessment of different solutions because of the iterative and repeated computation of critical paths (i.e., the longest path in a network). The graph method was used to carry out topology sorting one time, which increases computational efficiency significantly. Moreover, this research effort implemented parallel computing, provided an approximate Bayesian assessment of optimum GA parameters, and introduced an effective stopping criteria, all of which further improve computational efficiency. The methods provided here represent the groundwork for analyzing even larger networks, and can be extended to different construction engineering and management problems such as resource-constrained scheduling and resource leveling. Because the methodological improvements proposed here are non-domain-specific, they can be used to efficiently solve different computational problems.

Acknowledgments

Computational resources and services used in this work were provided by the High Performance Computing and Advanced Research Computing support (HPC-ARCs) Group of Queensland University of Technology, Brisbane, Australia. The authors also thank Dr. Justin Lee for his technical support in genetic algorithm design.

References

Affenzeller, M., Winkler, S., Wagner, S., and Beham, A. (2009). *Genetic algorithms and genetic programming*, CRC Press, Boca Raton, FL.

Alcaraz, J., and Maroto, C. (2001). "A robust genetic algorithm for resource allocation in project scheduling." *Ann. Oper. Res.*, 102(1–4), 83–109.

Amdahl, G. M. (1967). "Validity of the single processor approach to achieving large scale computing capabilities." *Proc., Spring Joint Computer Conf.*, Association for Computing Machinery, New York, 483–485.

Aminbakhsh, S., and Sonmez, R. (2016). "Discrete particle swarm optimization method for the large-scale discrete time-cost trade-off problem." *Expert Syst. Appl.*, 51, 177–185.

Boussad, I., Lepagnot, J., and Siarry, P. (2013). "A survey on optimization metaheuristics." *Inf. Sci.*, 237, 82–117.

Chassiakos, A. P., and Sakellaropoulos, S. P. (2005). "Time-cost optimization of construction projects with generalized activity constraints." *J. Constr. Eng. Manage.*, 10.1061/(ASCE)0733-9364(2005)131:10(1115), 1115–1124.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to algorithms*, MIT Press, Cambridge, MA.

Debels, D., and Vanhoucke, M. (2007). "A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem." *Oper. Res.*, 55(3), 457–469.

Dharwadker, A., and Pirzada, S. (2011). *Applications of graph theory*, CreateSpace Independent Publishing Platform, Gurgaon, Haryana, India.

Fang, C., and Wang, L. (2012). "An effective shuffled frog-leaping algorithm for resource-constrained project scheduling problem." *Comput. Oper. Res.*, 39(5), 890–901.

Feng, C.-W., Liu, L., and Burns, S. A. (1997). "Using genetic algorithms to solve construction time-cost trade-off problems." *J. Comput. Civ. Eng.*, 10.1061/(ASCE)0887-3801(1997)11:3(184), 184–189.

Floudas, C. A., and Pardalos, P. M. (2008). *Encyclopedia of optimization*, Vol. 1, Springer, Berlin.

Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M., and Gagné, C. (2012). "DEAP: Evolutionary algorithms made easy." *J. Mach. Learn. Res.*, 13, 2171–2175.

Gaglioppa, F., Miller, L. A., and Benjaafar, S. (2008). "Multitask and multistage production planning and scheduling for process industries." *Oper. Res.*, 56(4), 1010–1025.

Ghoddousi, P., Eshtehardian, E., Jooybanpour, S., and Javanmardi, A. (2013). "Multi-mode resource-constrained discrete time-cost-resource optimization in project scheduling using non-dominated sorting genetic algorithm." *Autom. Constr.*, 30, 216–227.

Hartmann, S., and Briskorn, D. (2010). "A survey of variants and extensions of the resource-constrained project scheduling problem." *Eur. J. Oper. Res.*, 207(1), 1–14.

He, Z., He, H., Liu, R., and Wang, N. (2017). "Variable neighbourhood search and tabu search for a discrete time/cost trade-off problem to minimize the maximal cash flow gap." *Comput. Oper. Res.*, 78, 564–577.

Hegazy, T. (1999). "Optimization of construction time-cost trade-off analysis using genetic algorithms." *Can. J. Civ. Eng.*, 26(6), 685–697.

Hold-Geoffroy, Y., Gagnon, O., and Parizeau, M. (2014). "Once you scoop, no need to fork." *Proc., 2014 Annual Conf. on Extreme Science and Engineering Discovery Environment*, Association for Computing Machinery, New York, 60.

Kahn, A. B. (1962). "Topological sorting of large networks." *Commun. ACM*, 5(11), 558–562.

Kandil, A., and El-Rayes, K. (2006). "Parallel genetic algorithms for optimizing resource utilization in large-scale construction projects." *J. Constr. Eng. Manage.*, 10.1061/(ASCE)0733-9364(2006)132:5(491), 491–498.

Kandil, A., El-Rayes, K., and El-Anwar, O. (2010). "Optimization research: Enhancing the robustness of large-scale multiobjective optimization in construction." *J. Constr. Eng. Manage.*, 10.1061/(ASCE)CO.1943-7862.0000140, 17–25.

Kim, J.-L. (2013). "Genetic algorithm stopping criteria for optimization of construction resource scheduling problems." *Constr. Manage. Econ.*, 31(1), 3–19.

Kim, J.-L., and Ellis, R. D., Jr. (2008). "Permutation-based elitist genetic algorithm for optimization of large-sized resource-constrained project scheduling." *J. Constr. Eng. Manage.*, 10.1061/(ASCE)0733-9364(2008)134:11(904), 904–913.

Knuth, D. E. (1968). *The art of computer programming*, Vol. 1, Addison-Wesley, Reading, MA.

Long, L. D., and Ohsato, A. (2009). "A genetic algorithm-based method for scheduling repetitive construction projects." *Auto. Constr.*, 18(4), 499–511.

Menesi, W., Golzarpoor, B., and Hegazy, T. (2013). "Fast and near-optimum schedule optimization for large-scale projects." *J. Constr. Eng. Manage.*, 10.1061/(ASCE)CO.1943-7862.0000722, 1117–1124.

Merkle, D., Middendorf, M., and Schmeck, H. (2002). "Ant colony optimization for resource-constrained project scheduling." *IEEE Trans. Evol. Comput.*, 6(4), 333–346.

Michalewicz, Z. (1999). *Genetic algorithms + data structures = evolution programs*, Springer, Berlin.

Monghasemi, S., Nikoo, M. R., Fasaee, M. A. K., and Adamowski, J. (2015). "A novel multi criteria decision making model for optimizing time-cost–quality trade-off problems in construction projects." *Expert Syst. Appl.*, 42(6), 3089–3104.

Python [Computer software]. Python Software Foundation, Wilmington, DE.

Red Hat Enterprise Linux version 6.4 [Computer software]. Red Hat, Inc., Raleigh, NC.

Senouci, A. B., and Eldin, N. N. (2004). "Use of genetic algorithms in resource scheduling of construction projects." *J. Constr. Eng. Manage.*, 10.1061/(ASCE)0733-9364(2004)130:6(869), 869–877.

Sonmez, R., and Bettemir, Ö. H. (2012). "A hybrid genetic algorithm for the discrete time-cost trade-off problem." *Expert Syst. Appl.*, 39(13), 11428–11434.

Sunnaker, M., Busetto, A. G., Numminen, E., Corander, J., Foll, M., and Dessimoz, C. (2013). "Approximate Bayesian computation." *PLoS Comput. Biol.*, 9(1), e1002803.

Tran, D.-H., Cheng, M.-Y., and Prayogo, D. (2016). "A novel multiple objective symbiotic organisms search (MOSOS) for time-cost–labor utilization tradeoff problem." *Knowledge-Based Syst.*, 94, 132–145.

Vanhoucke, M., and Debels, D. (2007). "The discrete time/cost trade-off problem: Extensions and heuristic procedures." *J. Scheduling*, 10(4), 311–326.

Zhang, H. (2011). "Ant colony optimization for multimode resource-constrained project scheduling." *J. Manage. Eng.*, 10.1061/(ASCE)ME.1943-5479.0000089, 150–159.

Zhang, H., Li, X., Li, H., and Huang, F. (2005). "Particle swarm optimization-based schemes for resource-constrained project scheduling." *Autom. Constr.*, 14(3), 393–404.

Zhang, Y., and Ng, S. T. (2012). "An ant colony system based decision support system for construction time-cost optimization." *J. Civil Eng. Manage.*, 18(4), 580–589.