Computers & Industrial Engineering 100 (2016) 52–57

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

Short Communication

A tabu search with gradual evolution process

Yoshinori Suzuki *, Juan David Cortes

Department of Supply Chain & Information Systems, College of Business, Iowa State University, 2340 Gerdin Business Building, Ames, IA 50011-1350, USA

A R T I C L E   I N F O

Article history:
Received 10 June 2016
Received in revised form 8 August 2016
Accepted 9 August 2016
Available online 10 August 2016

Keywords:
Optimization
Metaheuristic
Tabu search
Evolution
Virus

A B S T R A C T

We investigate a new framework for executing tabu search (TS). A unique aspect of this framework is that it performs multiple small-scale TS runs iteratively to identify the most promising area of the feasible region before executing the final TS run. The basic idea is inspired by recent viral transmission incidents, which showed that a virus would often go through an indirect and gradual evolution process to transform itself into a new form. Numerical experiments conducted with randomly-generated vehicle routing instances demonstrate interesting results.

1. Introduction

Tabu search (TS) is one of the most widely used metaheuristics in practice and academic research, especially for solving difficult combinatorial problems (e.g., traveling-salesman, vehicle routing, sequencing, network-design, and spanning tree problems) (Toth & Vigo, 2003). Past studies have investigated the question of how to improve the effectiveness (solution quality) and efficiency (solution speed) of TS by exploring a variety of diversification and intensification strategies, neighborhood definitions, tabu-list updating procedures, and termination criteria (Gendreau & Potvin, 2010; Ho, Yang, Ni, & Wong, 2001). To the best of our knowledge, however, none of these past studies have explored the question of how the TS performance can be improved by changing the nature (e.g., size) of the problem during the optimization run. Changing the problem characteristics during the optimization run can be potentially beneficial, as it can provide the information that would be difficult to obtain under the conventional TS execution method (where the nature of the problem is fixed), which may be used to enhance the TS performance. We investigate a new way of executing TS that changes the nature of the problem dynamically during the optimization run, and contrast its performance with that of a well-known TS procedure.

2. Basic idea and related literature

2.1. General idea

The basic idea of our approach (framework) is inspired by the viral transmission incidents observed between swine (pig) and avian (bird) flues several years ago. These incidents showed that while influenza A virus (bird flu) in its original form can hardly infect human beings, it can transform itself into a new form which can pose threats to humans if it goes through pigs. In other words, while moving directly from birds to humans is an infeasible path for the flu, moving indirectly from birds to humans through pigs is a feasible path (see Fig. 1). This means that, although a virus residing in one environment (E_1) cannot move to a very different environment (E_M) directly, it may be able to eventually transform itself into a new form that can reside in E_M if, for example, it goes through an indirect transmission path: $E_1 \to E_2 \to \ldots \to E_m \to E_{m+1} \to \ldots \to E_M$, where E_m and E_{m+1} represent different, but similar, living environments (hosts populations) for the virus. Our approach is based on this idea of sequential (and gradual) viral transmission and evolution.

2.2. Related literature

A limited number of studies have developed algorithms that involve the emulation of viral movements among host populations, most of which relied on the use of genetic algorithms (GAs). Arakawa, Kubota, and Fukuda (1996) implemented a virus-based GA to the trajectory generation problem, particularly with application to robotics. The algorithm divides the population into several subpopulations (virus and host populations), wherein each virus

* Corresponding author.
E-mail addresses: ysuzuki@iastate.edu (Y. Suzuki), davidco@iastate.edu (J.D. Cortes).

http://dx.doi.org/10.1016/j.cie.2016.08.004
0360-8352/© 2016 Elsevier Ltd. All rights reserved.

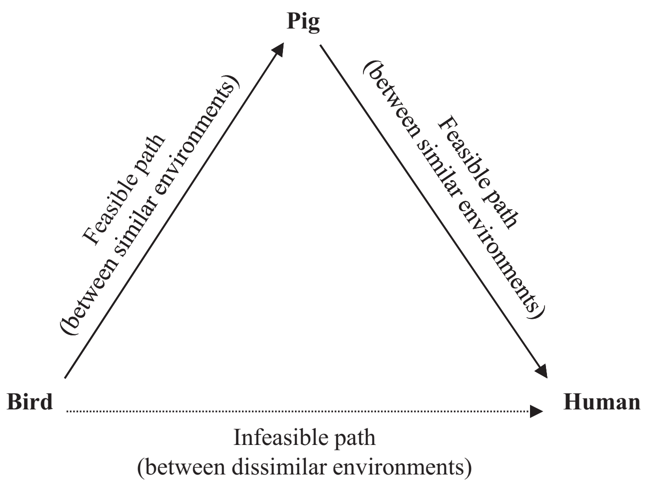
Pig
Bird
Human
Feasible path (between similar environments)
Feasible path (between similar environments)
Infeasible path (between dissimilar environments)

Fig. 1. Viral transmission paths.

subpopulation undergoes the processes of crossover, mutation, infection, selection, and cross-population migration. Chun, Jung, and Hahn (1998) developed and compared an Immune algorithm (IA), an evolution strategy (ES), and a GA. Their approach, which heavily relies on stochastic transition and movements of viruses, is based on the genetic abilities of viruses to cope with invading antigens and the production of antibodies to exclude the antigens. Cortes, Onieva, Munuzuri, and Guadix (2013) developed a virus-inspired algorithm, designed specifically for the elevator dispatching problem, in which each cell may be infected with a virus based on a probability, where the infection level of each cell depends on a given antigen level.

Some studies developed virus-like algorithms specifically for routing problems. Kanoh and Tsukahara (2010) proposed a variant of a GA for the capacitated vehicle routing problem (CVRP) which updates and maintains a population of k viruses (each of which is a single route) with a better likelihood of producing quality feasible solutions. Each of these k viruses has the ability to infect solutions to make part of them resemble the virus, so that, after several iterations, the method leads to improved solutions. Prins (2004) presented a viral-movement like hybrid GA for the distance-constrained CVRP that initially solves a problem by relaxing the vehicle capacity and maximum tour length constraints, and subsequently executes a second stage optimization run consisting of an optimal splitting procedure to generate competitive solutions with feasible tours. Suryadi and Kandi (2012) developed a viral systems algorithm emulating the cellular infection process to solve the traveling salesman problem. They proposed two moves (i.e. lytic replication and lysogenic replication) to change the locations (i.e., sequence) of the nodes within the solution.

While valuable in many respects, none of the above studies has developed a framework specifically for TS that mimics the viral infection process, nor proposed a framework that solves a given problem by emulating an indirect, gradual transformation process of certain flu viruses. This study fills this gap in the literature by investigating a new framework that combines TS and the aforementioned gradual viral transformation process. Given the widespread use of TS among researchers and practitioners, there should be considerable value in developing such a framework.

3. The framework

3.1. Solution strategy

First, we create a series of problems $\{P_1, P_2, P_3, \ldots, P_m, \ldots, P_M\}$, where P_M is the focal (original) problem to be solved (which is difficult to solve) and P_1, P_2, ... represent the problems similar to, but less complex than, P_M (their problem size, denoted n, is smaller than that of P_M). The problems are numbered such that P_m has a smaller n than P_{m+1}. n of two adjacent problems (P_m and P_{m+1}) are similar, but those of P_1 and P_M are substantially different (i.e., P_1 is substantially easier to solve than P_M). Second, we solve P_1 to either optimality or near optimality (given that P_1 is easy to solve, this can be done, for example, by using enumeration). We denote the best solution found for P_1 as S_1. Third, we solve P_2 via the method that can perform an in-depth neighborhood search (TS) by using S_1 (or its slightly modified version) *as the initial solution.* Fourth, we repeat the above process (i.e., solve P_m by running TS which uses S_{m-1} as the initial solution) many times, successively moving from one problem to another, until we reach and solve the focal problem P_M.[1]

Note that, in essence, our framework performs the following set of actions: (i) generate an optimal (or near-optimal) solution to P_1, the simplified form of P_M (denoted S_1), (ii) transmit S_1 to a similar, but more complex, environment (P_2) and let it evolve within the new environment until it transforms itself into another form that performs well within this new environment (S_2), and (iii) repeat this transmission (evolution) process many times until the solution transforms itself into a form that performs well in the final environment (S_M) (see Fig. 2 for the outline of our framework). Also note that, although executing the above framework may seem time consuming, as it requires a series of TS runs, its actual run time may not be very long for two reasons. First, in most TS runs n is much smaller than that of P_M, so that these runs would not take long time. Second, since P_{m-1} and P_m are "similar" problems, meaning that the best solution found for P_{m-1} is a good candidate for the optimal solution of P_m, the TS run for P_m (which uses a proxy of the P_{m-1} best solution as the initial solution) should, in many cases, converge to quality solutions quickly.[2]

3.2. Diversification and intensification

Although in the previous paragraph we argued that each TS run is expected to find quality solutions quickly in the vicinity of the initial solution, this does not mean that we merely focus on searching the areas near the initial solution in each TS run. Since the initial solution may not always be located near the optimal solution, and since it is known that diversification is a useful strategy for finding quality solutions, we perform multiple diversifications in each TS run. However, because executing many diversifications in each TS run can be time-consuming, we employ an approach which performs intensive (a larger number of) diversifications when n is small, but performs a limited number of diversifications as n grows larger. There are two reasons for this.

First, from the standpoint of finding quality solutions, we may not need to perform as many diversifications when m is large (i.e., when n is large) as when m is small (when n is small). Note that, given that the change in n from P_m to P_{m+1} must always be "gradual" (i.e., the difference in n between P_m and P_{m+1} is small;

[1] Strictly speaking, the sequence $P_1, \ldots, P_m, \ldots, P_M$ needs not be ascending in problem complexity (i.e., problem size needs not grow with m). The sequence can also be descending in problem complexity (i.e., evolutionary process can start with a more complex problem and use its best solution as the initial solution to solve the next, less complex, problem). This latter approach, however, would generally be difficult to implement because of higher computational requirements.

[2] Though our approach may look similar to dynamic programming (DP), they are fundamentally different. In DP, the original problem is divided into multiple *non-overlapping* sub-problems (stages), each of which is solved to determine the best solution for a multistage process. Our approach, in contrast, does not divide the original problem into stages, but instead creates multiple *overlapping*, interconnected problems (each of which is a reduced form representation of the original problem) and solves them sequentially to obtain a good candidate initial solution for the original problem.