SOCIAL STRUCTURE AND COLLECTIVE INTELLIGENCE IN PROBABILITY-BASED PARTICLE SWARM OPTIMIZATION FOR THE FOREST PLANNING PROBLEM

by

ANGELA TSAO

(Under the Direction of Pete Bettinger)

Forest planning can be represented as a type of spatially constrained combinatorial optimization. Many complex forest planning problems are computationally intractable, but metaheuristics allow for efficient discovery of high quality solutions. This project introduces a new velocity update procedure for a probability-based variant of the Particle Swarm Optimization algorithm, incorporating models of baboon information-pooling behavior to inform social influence among particles. The new baboon-based algorithm is tested over three forest planning problems in comparison to existing optimization strategies. The baboon communication mechanism significantly improves performance of the PSO on complex, higher-dimensional problems. We implement strategies for improving performance of probability-based optimization over a constrained search space and test the effect of different frameworks of social influence on algorithm performance.

INDEX WORDS: Combinatorial optimization, Constrained optimization, Probability optimization, Discrete particle swarm optimization, Forest planning, Nature-inspired optimization, Collective intelligence, Social networks

SOCIAL STRUCTURE AND COLLECTIVE INTELLIGENCE IN PROBABILITY-BASED PARTICLE SWARM OPTIMIZATION FOR THE FOREST PLANNING PROBLEM

by

Angela Tsao

BS, University of Georgia, 2021

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2021

SOCIAL STRUCTURE AND COLLECTIVE INTELLIGENCE IN PROBABILITY-BASED PARTICLE SWARM OPTIMIZATION FOR THE FOREST PLANNING PROBLEM

by

ANGELA TSAO

Major Professor:	Pete Bettinger
Committee:	Frederick Maier
	Chris Cieszewski

Electronic Version Approved:

Ron Walcott
Vice Provost for Graduate Education and Dean of the Graduate School
The University of Georgia
May 2021

ACKNOWLEDGEMENTS

I would like to express my gratitude to my advisor, Dr. Bettinger, for his guidance and support of my foray into the study of heuristic optimization algorithms. His Advanced Forest Planning course inspired me to explore this subject deeper and incorporate my experiences in ecology with my interest in computational intelligence. I thank him for his continued mentorship, especially in talking through ideas and sharing patient feedback over numerous revisions.

I would also like to thank Dr. Maier and Dr. Cieszewski for their support and being members of my thesis committee. Their comments and advice helped shape the direction of this final product.

Finally, I am grateful to my family for their unwavering love.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

In natural resource management, forest planning involves decision-making about when and where to conduct various forest management activities. Forest planning can occur at different scales and involves selection of appropriate management prescriptions at certain locations over a time horizon, with a goal of achieving stakeholder-defined objectives. These goals may be as diverse as economic production, environmental services, or social good. Due to the combinatorial complexity of potential management configurations, mathematical programming methods like linear programming and mixed integer programming have been applied in the development of forest plans. However, the size or nature of different forest planning problems may render these methods computationally inefficient or infeasible for practical use. Some researchers have applied search and optimization heuristics to forest planning in an attempt to produce high-quality forest plans more efficiently (Bettinger et al., 2009); promising results from application of heuristic algorithms like Simulated Annealing and Tabu Search (Borges & Eid, 2014; Bettinger et al., 2009) suggest that the development of efficient, robust heuristics is important for the forest planning problem.

Particle Swarm Optimization (PSO) is a population-based optimization algorithm based on the behavior of interacting individuals in a swarm. In the canonical PSO, candidate solutions represent individual particles in a larger swarm. Each particle possesses a position, a velocity, and memory of its personal best position ever found as well as the swarm's best position ever found (Eberhart & Kennedy, 1995). While PSO achieved much success in many different types

of problem (particularly over the continuous domains it was originally designed for), its discrete variants did not perform particularly well when applied to the forest planning problem. The Roulette Wheel PSO (RWPSO) is a variant of PSO designed specifically to perform well on nominal variable problems like the forest planning problem. While RWPSO outperformed existing heuristics on two harvest scheduling problems, it had inferior performance on a higher-dimensional, more complex harvest scheduling problem (Smythe, 2012) and struggles with constraint-handling. PSO-class algorithms lack native constraint-handling, so algorithms must use problem-level methods that may transfer poorly over probability-based movement. Despite the initial promising results of RWPSO, there has been a lack of further research into these challenges and the broader potential of probability-based PSO variants for solving the forest planning problem. Our project extends this work by updating the RWPSO algorithm with behavioral models of social structure from cognitive ecology.

We introduce a direct information-sharing behavior based on baboon social groups to the RWPSO algorithm; a new velocity update function is developed that incorporates information from both global and neighborhood-level sources, modeled on baboon social structure. In addition, the efficacy of the baboon modification is tested across various network structures to evaluate their impact in the harvest scheduling domain. We incorporate an embedded constraint handling method for operation over a discrete, nominal search space, which coordinates the probability-space flight of RWPSO with local movement in the solution space. In testing over three test cases, B-RWPSO matched the best solution found by mixed integer programming in two low-dimension test cases and found a solution within 2.5% of the mixed integer solution on the most complicated test forest. B-RWPSO improves upon the performance of the original RWPSO. Paired testing to compare the isolated effect of the baboon modification shows that B-

RWPSO improves the RWPSO's mean best objective function by a range of 10.4% to 17.4%, depending on parameter settings. Our testing of velocity initialization strategies also provides insight into the interaction between spatial adjacency handling and population evolution in generational algorithms.

Chapter 2 of this work summarizes existing background information about the forest planning problem domain. Details are also given about the canonical particle swarm optimization algorithms and modifications that have improved performance, including the probability-based discrete RWPSO variant. Chapter 2 also briefly discusses collective intelligence, and how insights from this field of behavioral ecology can, like its swarm intelligence subfield, augment development efforts in nature-inspired algorithms. Chapter 3 describes the details of design and development of the Baboon-Modified RWPSO (B-RWPSO). Chapter 4 presents the results of testing B-RWPSO in comparison to existing heuristics and mixed integer programming on three spatial harvest scheduling problems. We assess different parameter combinations and discuss the impact of constraint-handling. In addition, testing is conducted to evaluate performance of the baboon-based information-pooling process in various network topology types. Finally, Chapter 5 provides a summary discussion of this project and directions for future work.

CHAPTER 2

BACKGROUND

2.1 FOREST PLANNING

Forest planning is a field of natural resource management that has a long tradition of using mathematical programming for harvest scheduling. Forest planning efforts often involve selecting the timing and location of forest management activities, at the forest or landscape level, to best meet the objectives of the landowner or land managers. This type of problem can be represented as constrained optimization, and is inherently an allocation problem (where to go and what management activity to apply) under graph-coloring adjacency constraints. In forestry, a management plan could be designed to maximize an economic (e.g., net present value), commodity production (e.g., timber volume), environmental (e.g., wildlife habitat), or social (e.g., net human benefit, jobs, etc.) objective. In addition, a management plan could be designed to minimize environmental damage, management costs, and other measures of outcomes from the assignment of activities to a landscape; the quality of forest plans can be assessed based on the calculated expected outcomes. Common constraints in forest management plans include those applied to the forest inventory (e.g., to prevent depletion), to the timing and placement of certain activities (e.g., to prevent clearcuts from becoming too large), to the budgets that are assumed, and to many other economic, environmental, and social concerns as long as they can be quantified.

However, when the number of decision variables increases beyond trivial levels, a planning problem suffers combinatorial explosion— also known as the *curse of dimensionality*— where the number of distinct forest plans that can be developed is:

$$(\text{Number of decision variables related to a stand})^{\text{number of stands}}$$

rendering the problem basically intractable to the human mind. Linear programming (LP) was the first quantitative method applied in the development of forest plans for areas larger than small forests. Linear programming utilizes the Simplex method to solve a problem that is arranged in a detached coefficient matrix. LP remains a common optimization technique in forest planning, but assumes that each decision variable can be assigned a continuous real number (the assumption of divisibility). In practical terms, linear programming may produce a solution that splits a stand's timber harvest amongst two or more time periods. An alternative strategy, the mixed integer programming model, forces all decisions into integer or discrete solution values and requires the use of branch and bound (Lawler & Wood, 1966), cutting plane, or other methods. In contemporary forest planning efforts, there is a need for integer solution values for some decision variables, to control the timing and size of final harvests (clearcuts) or the size and location of wildlife habitat patches. Depending on the size of the problem and the nature of the data to which the problem is applied, these methods may be computationally intensive and require significant time to arrive at the optimal solution. Heuristics have thus been suggested as alternatives to mixed integer programming. While heuristics can be designed to produce high-quality solutions (forest plans), they cannot guarantee optimal solutions will be located. Additionally, many heuristic algorithms have been primarily designed for deployment over unconstrained optimization problems, so constraint-handling methods must be integrated into the algorithm to solve certain constrained problems and keep solutions in feasible regions (Liu &

Wang, 2019). Therefore, the development of computationally efficient and robust heuristic methods is important in the forest planning problem.

2.1.1 Harvest Scheduling

Harvest scheduling as described by Bettinger & Zhu (2006) presents a simplified version of the forest planning problem, narrowed down to focus primarily on the single objective of timber harvest volume. In this version of the harvest scheduling challenge, an individual stand in the forest is scheduled either with a single time period to be harvested or left untouched throughout the management horizon; however, each stand may only be cut once. To simplify the representation of this challenge, the set of "management prescriptions" or potential locations at a dimension in the vector space is limited to just clearcuts of an entire stand. Real-world considerations would allow for alternative management activities such as partial harvests or thinnings. The challenge is one of combinatorial optimization with graph-coloring constraints, as different stands (spatial units) within a forest need to be harvested at appropriate times to maintain even-flow cutting while subject to *unit-restriction* spatial adjacency constraints (like those found in a typical graph-coloring problem).

Specifically, we model the problem such that a stand within a forest may not be clear-cut in the same time period as an adjacent stand, and depending on the type of tree grown, a stand may not be harvested until reaching a certain age. These spatially-based harvest constraints represent environmental protections during forest green-up periods. One unique consideration for the spatial constraints used in forest planning is the disanalogy between spatial relationships in the physical forest and the numerical representation of stands (Figure 2) used by a heuristic. For example, two stands that are 'neighboring' in the vector representation of a forest plan (e.g.

stands 3 and 4 in the 40-stand forest, Figure 2) may not be subject to adjacency constraints because they map to real forest stands that are far apart. Unlike traditional ordinal problems, constraints lack shape or order in the solution space and reflect the combinatorial relationship between stand-level assignments in a forest plan.

In order to maximize even-flow of harvest volume, we then define the objective function:

$$f_1 = \sum_{k=1}^{z} \left(T - \sum_{n=1}^{d} a_n h_{n,k} \right)^2$$

Where for each simulated forest plan, we sum the time period's squared deviations of scheduled harvest volume from target harvest volume in each period. T is the target harvest volume for each time period, a_n is the acres in stand n, $h_{n,k}$ is the volume harvested per acre in stand n for time period k, and d is the total number of stands. z represents the number of time periods. Here, f_1 represents the squared error from a determined target harvest volume, an optimal schedule is one that minimizes f_1.

Optimization heuristics may be used to search potential configurations of forest plans for an optimal or near-optimal solution by representing the problem as an n-dimensional search space, where each dimension n can have a value (a harvest time period or management prescription) from the set of allowable k. This project considers three different forests to which an array of different optimization strategies have previously been employed. These forests include a 40-stand northern forest (Bettinger & Zhu, 2006; Smythe, 2012), a 73-stand Daniel Pickett Forest (Bettinger & Zhu, 2006; Smythe, 2012), and a 625-stand southern forest (Bettinger & Zhu, 2006; Smythe, 2012).

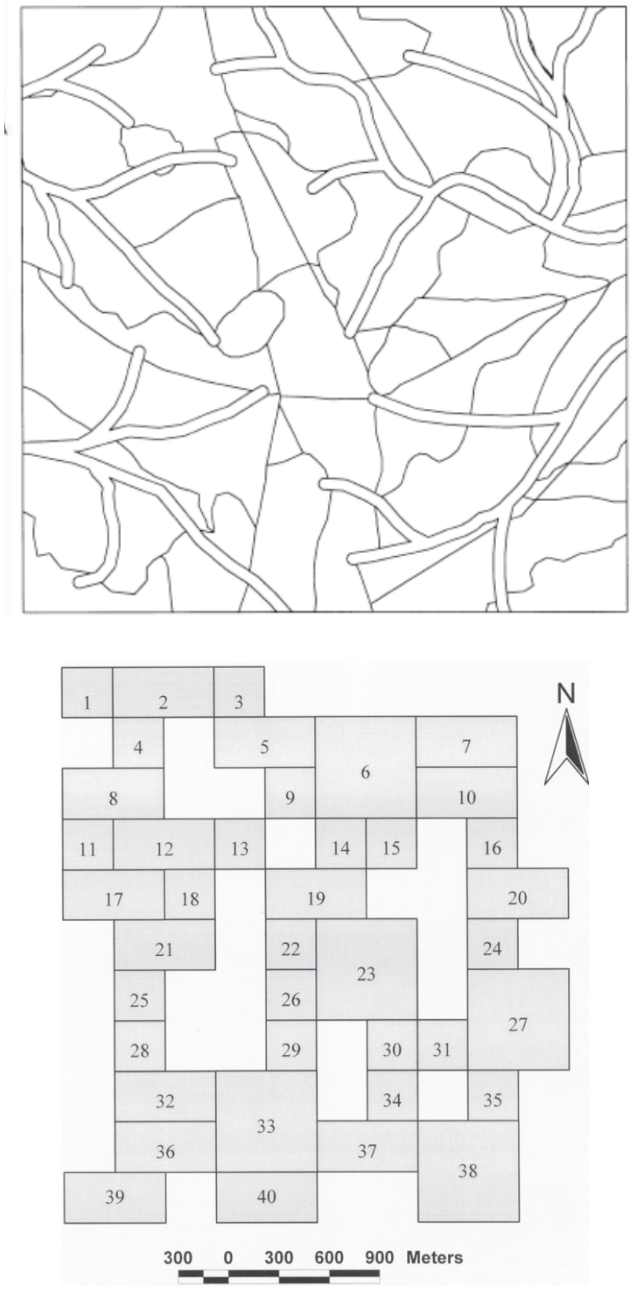
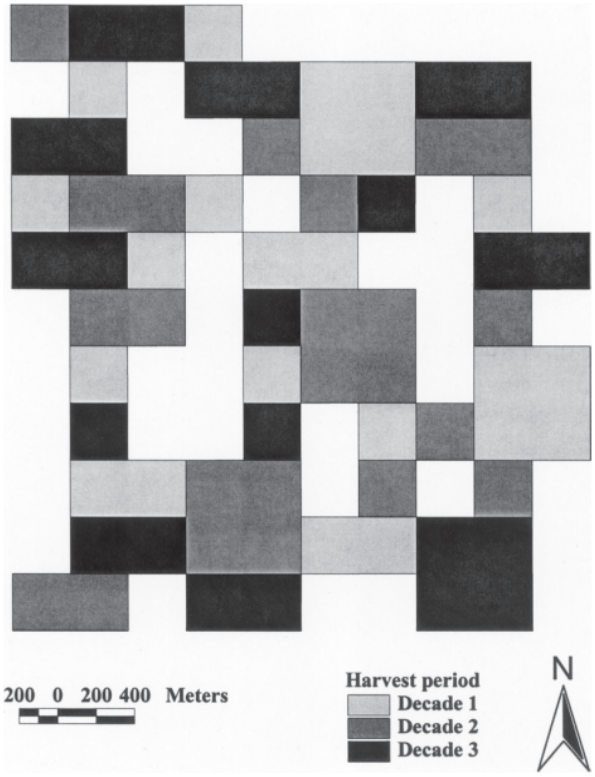


Figure 1. The 73-stand forest (left) and the 40-unit northern forest (right)

These forest planning problems span three time periods, giving each stand a choice among 4 management options (time period 1, 2, or 3, or no cut at all). The target harvest and projected yields per time period are determined *a priori* and input into the problem. For the 40-stand forest the target harvest is 9,134.6 m^3, for the 73-stand forest the target harvest is 34,467

MBF (thousand board feet), and for the 625-stand forest the target harvest is 2,972,462 tons (Bettinger & Zhu, 2006). The target volumes for these forests are the derived linear programming solutions; however, because real-world constraints on harvest require that entire stands be managed as a unit, the optimal solution derived from a forest plan must be found with methods that accommodate this requirement. Therefore, mixed-integer solutions are used as a point of comparison in Chapter 4.



Position Vector: [2,3,1,1,3,1,3,3,2,2,1,2,1,2,3,1,3,1,1,3,2,3,2,2,1,1,1,3,3,1,2,1,2,2,2,3,1,3,2,3]

Figure 2: Representation of spatial adjacency constraints (unit-restriction model): forest plan spatial overlay vs. vector representation in harvest scheduling

2.2 PARTICLE SWARM OPTIMIZATION

Particle swarm optimization (PSO) is a population-based search algorithm modeled after the swarm behavior of animals such as birds and fish. The population in a PSO is referred to as

the swarm, while individuals in the population are particles in the swarm (Eberhart & Kennedy, 1995). Particles—or candidate solutions—represent a point or position in an *i*-dimensional search space, and they move to different positions in the search space based on their recorded velocity, which is in turn updated based on the local and global best locations in memory. In the forest planning context, each particle is a complete but mutable forest plan, and it possesses a current location, velocity, and memory of both its own best past location and the swarm-level best past location.

The population is initialized by generating randomly positioned individuals with random starting velocities. Like evolutionary algorithms, SI algorithms iteratively update individuals in the population across time-steps, which may be referred to as generations, epochs, or iterations. In an iteration of the algorithm for each particle's dimension *i*, the particle updates its position *x* based on its velocity and previous position. Meanwhile, it updates its velocity based on its previous velocity, its personal best position in memory *p,* and the swarm's best position in memory *g* (where "best" is determined by a user-defined objective function). For the velocity of dimension *i* at iteration *t*, the particle's previous velocity is multiplied by an inertia factor μ and the relative weight given to knowledge about the particle's local best and the swarm's global best positions is dependent on the cognitive influence factor, *c1*, and the social influence factor, *c2*. Stochasticity is introduced via random coefficients *r1* and *r2*:

$$v_i(t) = \mu v_i(t-1) + r_1 c_1\big(p_i - x_i(t-1)\big) + r_2 c_2\big(g_i - x_i(t-1)\big)$$

$$x_i(t) = x_i(t-1) + v_i(t)$$

Self-organization in the group arises from individuals' iterative velocity updates where aggregate movements result in superior movements as a manifestation of "swarm intelligence." The *many-wrongs principle* hypothesizes that group cohesion works to suppress the numerous individual

errors made by members of a group; in PSO, similarly, individual particles contribute to stochastic exploration but accelerate toward the best solution found by the swarm.

An oft-cited advantage of PSO is that its nature-inspired velocity mechanism removes the need for a gradient, thus allowing PSO to be applied on problems with objectives that are not differentiable. However, due to its modelling of velocities as a combination of direction and magnitude, there are difficulties in representing movement of particles in the solution space of a problem with nominal variables, such as the forest planning problem. Smythe (2012) notes that PSO is frequently adapted to travel in the probability space of nominal variables instead of their value space, as the latter lacks congruence with particles' movement orientation and magnitude. Researchers have made many iterative improvements to PSO (Freitas, 2020); this background section describes some modifications and variants of the PSO that show promise for the forest planning problem.

2.2.1 Social Interaction in PSO:

The standard PSO algorithm described above uses a singly informed network influence type with *gbest* topology, which means that all individual particles in the swarm receive social influence exclusively from the global leader ("exclusively" because the swarm is a singly informed network type; "from the global/swarm-level leader" because of *gbest* sociometry). In such a case, each particle is connected to every other one, in a *gbest* sociometry (although in any given iteration, only one social neighbor's influence is considered). However, it has been proposed that this *gbest* topology may be poorly suited for problems without long gradients in the function landscape (Kennedy & Mendes, 2006). The alternative *lbest* topology with "best-of-neighborhood" network type connects each particle only to its *k* immediate neighbors and

updates each particle's velocity based on the position of its best neighbor in the set of *k* nearest. This has been shown to curb premature convergence in PSO, as global leaders stuck in a local optimum cannot pull all other particles toward the local optimum. In contrast, the *gbest* "best of neighborhood" PSO has performed especially poorly on complicated, higher dimensional problems (Parsopolous & Vrahatis, 2005) where particles can easily fall into local optima.

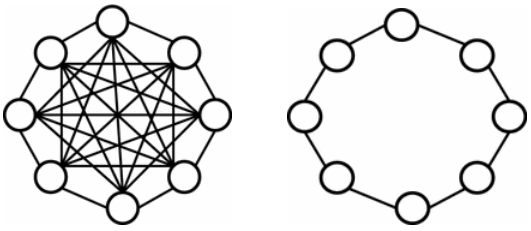


Figure 3. Network depiction of *gbest* topology (left) and *lbest* topology with k= 2 (right)

The neighborhood topology refers to *which other particles a given particle may ever see or interact with*. On the other hand, the network influence type (e.g., fully informed, singly informed/"best-of-neighborhood") refers to a particle's *willingness to use information from its visible neighbors*—the particle may use only the "best-of-neighborhood," or it could even receive social influence from every particle in its neighborhood. To improve upon these extremes, the Unified Particle Swarm Optimization (UPSO) was introduced to combine the *gbest* and *lbest* topologies by weighting each approach with a unification factor (Parsopoulos & Vrahatis, 2005). Kennedy and Mendes (2006) tested the performance of the fully informed particle swarm (FIPS), a network influence type of the PSO where each particle is influenced by the success of all of its neighbors (as opposed to just the global or neighborhood leader) and find that the FIPS outperforms the singly informed canonical PSO in some cases, depending on neighborhood topology (e.g., *gbest* vs. *lbest*). The importance of neighborhood topology is

evidenced by empirical data that particles need not be informed exclusively by the best performing particle in the swarm (Kennedy & Mendes, 2006). However, previous applications of PSO variants to the forest planning problem have not explored the effect of topology on problem-specific performance. Variation in the information influence structure and neighborhood topology can have a significant effect on convergence speed and likelihood of premature convergence (Vasquez, 2014), although these effects vary by problem and domain (Freitas, 2020). Our work in collective intelligence communicative procedures is inspired by these inquiries into the role of network influence type and neighborhood topology on algorithm performance.

2.2.2 Variable Parameters in PSO

The parameters in PSO include inertia weight μ, besides cognitive/social influence factors and two random coefficients. In the canonical PSO, these parameters are static throughout execution. The values of these parameters are critical to the algorithm's performance, as they help to modulate a balance between global exploration and local exploitation of the search space. Cognitive and social influence factors contribute to these two aims, respectively, while inertia helps balance the local and global search by weighting the value of previous particle trajectory (Shi & Eberhart, 1999). Some degree of parameter tuning is crucial to ensure that the swarm balances both exploration and exploitation. In particular, Eberhart and Kennedy (1995) found that a high cognitive influence factor can result in meandering around the search space, while a high social component could lead particles in the swarm to premature convergence at a local optimum.

Later studies by Shi and Eberhart (1999) then demonstrated an improvement to PSO performance when using a linearly varying inertia factor (within a user-specified range for inertia, between maximum start μ_1 and minimum ending value μ_2). The implementation of time-varying acceleration coefficients (social and cognitive factors) can also improve PSO performance (Ratnaweera, 2004).

2.2.3 Constrained Optimization with PSO

Although PSO is incredibly versatile and can be applied to a large number of domains, it lacks a native constraint handling strategy for adapting to constrained optimization challenges (like forest planning). The three main applicable strategies have been the penalty function method, feasibility-based rules method, and the constraint-preserving method (Sun, 2011). The former involves representing a constrained optimization problem as an unconstrained optimization problem, but modifying the objective function to include a penalty for infeasible solutions. A simple brute-force strategy adopted for PSO is the Preservation of Feasible Solutions Method, in which all feasible solutions found in the search space are preserved, and the optimal solution from this pool is selected after reaching stopping criteria (Hu & Eberhart, 2002). However, for domains like forest planning, checking feasibility for each individual in a swarm at every iteration to calculate a penalty assignment can become the most computationally expensive aspect of the algorithm. Many heuristics reduce the required number of feasibility checks by employing a "fly-back" type of constraint-preserving method by only accepting feasible moves to retain legal solutions at all times. Constraint-checking can then be streamlined by only checking adjacencies for the few stands affected by a move from an existing solution, and illegal moves are simply never made. If, however, a better feasible solution is separated from a current

particle by an infeasible intermediate move, then "islands" in the search space may not be found under the constraint-preserving method. Existing algorithms like the NVPSO and IVPSO work in continuous spaces to alter particle velocities when they move out of feasible regions (Sun, 2011). This vector-based acceleration augments the exploratory capabilities of the swarm, unlike earlier methods that used the "fly-back" protocol to undo particle's movement out the boundary of the occupied feasible zone (Sun, 2011). However, a clear extension of this acceleration strategy to probability-based PSO variants does not exist, as probabilities in the velocity domain map to likelihood of a location *k* at dimension *i* being selected, rather than mapping to a real value in the solution space.

2.3 ROULETTE WHEEL PSO

2.3.1 Discrete PSO:

Discrete PSO is a subtype of PSO that operates over a discrete search space. Forest planning is one type of problem that requires use of DPSO, as the categorical variables do not map coherently to a continuous space. In DPSO with interval or ordinal variables, the traditional equation for velocity update and position update can still be used (with a rounding modification), but baseline performance with DPSO is subpar over nominal-type variables. In such problems, locations or positions in the solution space are not ordered and lack spatial relationships to each other.

2.3.2 Roulette Wheel PSO:

RWPSO is a discrete multi-valued PSO designed specifically for use with nominal or categorical variables (Smythe, 2012), as in the case of harvest time periods scheduled in the

forest planning context. Because RWPSO is a probability-based variant, particles in the swarm travel over a probability space that maps to locations in the solution space. The RWPSO includes four static parameters that have been adapted to explicitly adjust roulette wheel values, namely stopping criteria (number of iterations to run), swarm size, maximum step size (of a change in probability), and social emphasis. An additional fifth parameter, cognitive emphasis, is simply equal to 1-social emphasis.

In RWPSO, every potential location *k* in dimension *i* of a particle *n* in the swarm is assigned a roulette wheel probability that gets updated at each iteration *t* (referred to as a velocity $v_{i,k}(t)$) (Smythe, 2012). Because these velocities are based on a roulette wheel, the velocities $v_{i,k}(t)$ for all locations *k* in the dimension sum to 1. The original RWPSO initializes starting velocities uniformly at each location *k* for each dimension *i* to the reciprocal of the number of permissible locations in dimension *i.* The velocity update equation parallels that of standard PSO:

$$v_{i,k}(t) = v_{i,k}(t-1) + m\left(s\big(B(g_i,k) - B(x_i(t-1),k)\big) + (1-s)\big(B(p_i,k) - B(x_i(t-1),k)\big)\right)$$

$$\text{where } B(x_i,k) = \begin{cases} 1 & if\ x_i = k \\ 0 & else \end{cases}$$

Given updated velocities, a particle calculates its new position for each dimension *i* by selecting a location for *i* in accordance with the probability distribution given by the roulette wheel probability of locations in each dimension.

$$P(x_i(t) = k) = \frac{v_{i,k}(t)}{\sum_{a=1}^{4} v_{i,a}(t)}$$

RWPSO has been tested on the forest planning problems in (Bettinger & Zhu, 2006; Smythe, 2012), and has achieved performance equal or superior to existing metaheuristics on

certain types of problems. Notably, RWPSO was superior to the forestry domain-specific Raindrop Optimization algorithm and existing heuristic methods like Tabu Search and Threshold Accepting when applied to the 40-stand and 73-stand forests (Bettinger & Zhu, 2006; Smythe, 2012). These promising results suggest that the RWPSO may be advantageous in forest planning. However, the RWPSO was inferior to both RO and Threshold Accepting on the 625-stand forest, and seemed to converge prematurely toward local optima (Smythe, 2012). This poor performance, and the comparative advantage of Raindrop Optimization algorithm, may be related to incompatibilities between the structure of the solution space, regions of infeasibility, and RWPSO's constraint handling. RWPSO thus requires additional refining and modifications that can enhance social communication. RWPSO also adds a multiplicative layer of time complexity in that velocity and position updates must be calculated over t iterations for n particles across i dimensions and k potential locations. The time complexity of RWPSO increases dramatically with combinatorial complexity for problems, especially those with many dimensions or many potential decisions in a dimension. To mitigate this challenge, we propose the baboon social structure modification as a way of updating information pooling in addition to improving the time complexity of the algorithm.

2.4 COLLECTIVE INTELLIGENCE

In empirical studies from cognitive ecology, it has been observed that group decisions generally improve in accuracy with an increase in the number of individuals involved in the decision-making process (Santos & Przybyzin, 2016). Collective intelligence may arise in intelligent species via a centralized social structure regulating the process of information pooling. Swarm intelligence is a lower-level manifestation of CI and a more decentralized information-

sharing process, including simple mechanisms like stigmergy (Krause, 2010). SI is what algorithms like PSO, ant colony optimization, and bee colony algorithm use; it differs from CI in that SI is the individual decisions of interacting organisms in the group, while CI refers to the group-level decision-making process of higher-level organisms. A swarm or group achieves CI via mechanisms of social structure and hierarchical communication. Within the CI-based baboon algorithm, we test an empirical model of baboon behavior as described by Strandburg-Peshkin & Farine (2015).

Collective Intelligence has the potential to complement the social-psychological phenomena built into PSO. Because CI governs centralized interactions between group members, more complicated relationships and patterns of social influence can be built in with structural approaches. These mechanisms may provide more nuanced communication between individuals, mitigating some issues in PSO related to premature convergence. Kennedy (2000) built a spatial *lbest* neighborhood topology based on clustering with "social stereotyping," simulating the sociological process of group assimilation and identity formation. This modification sets an example for how insights from CI may be integrated into the PSO as a behavioral mechanism or modification of the velocity update function.

CHAPTER 3

ALGORITHM DEVELOPMENT

3.1 B-RWPSO (BABOON-BASED ROULETTE WHEEL PARTICLE SWARM OPTIMIZATION)

The baboon modification introduces social structure to the Particle Swarm Optimization's standard particle-based velocity update mechanism, where traditionally the particle's velocity is updated based on cognitive and social coefficients, the population's global best position in memory, and the particle's own individual best position in memory. In the baboon algorithm, some particles directly update their velocity based on a collective intelligence approach that accounts for the local exploration of "movement initiators" in an iteration. Additionally, many of the improvements to the canonical PSO (discussed in Section 2.2) do not translate over to probability-based discrete PSO; certainly, the reframing of parameters as roulette wheel probabilities in RWPSO precludes certain strategies from being logically extended from PSO. As such, we adapt some promising features from the continuous PSO literature— like TVAC and innovative constraint-handling— and additionally develop new modifications to work specifically for an algorithm traveling over the probability space instead of the solution space.

3.1.1 Baboon Information-Pooling Behavior

Baboon troops interact with extreme egalitarianism when foraging (Strandburg-Peshkin & Farine, 2015), which allows for robust global exploration. There is also strong preference in baboons for moving toward locations that have previously been occupied by other baboons in the

troop (Strandburg-Peshkin, 2017), which supports a social intelligence ecosystem for search intensification. In wild movements, a random number of baboons will choose to move at any given moment, becoming "movement initiators" for a period of time. Surprisingly, these baboons need not have any social status or standing in the group; any baboon in the troop can initiate a movement and successfully influence others. Non-initiator baboons that decide where to move based on the decisions and distribution of these initiators are known as "followers".

An algorithm is implemented that randomly selects between 2.5% and 10% of baboons in the population to be movement initiators in an iteration. The B-RWPSO uses the constraint-preserving method (Sun, 2011) as the means of constraint handling and restricting search to only feasible solutions/forest plans. Baboons in an iteration are subject to strict constraint-preserving with fly-back implemented in the position function (which assigns a position based on probabilities specified by the individual's roulette wheel "velocities"). Based on a decision-making function modelled by ecologists with individual-level tracking data (Strandburg-Peshkin & Farine, 2015), follower baboons copy the velocity update (change in velocity) of movement initiators in their static social neighborhood. However, if the initiators branch in different directions, the followers may choose to compromise and take the path in between two initiating subgroups. Thus, there emerge 3 potential behavioral patterns that can be followed by a baboon in any iteration.

1) a) Randomly chosen baboon movement initiators OR b) baboons without any initiators in their social neighborhood both use the standard velocity update equation to calculate their new velocities
2) Follower baboons with exactly one initiator located in their social neighborhood
3) Follower baboons with more than one initiator located in their social neighborhood

Categories 1a and 1b follow the standard RWPSO update equation discussed in Section 2.3. However, follower baboons (categories 2 and 3) use two different behavioral equations, depending on how many initiators moved in their neighborhood in any given iteration. In the case of neighborhoods with a single initiator, all followers in the neighborhood will copy the change in velocity of the movement initiator (category 2). This equation can be represented most simply by:

$$v_{i,k}(t) = v_{i,k}(t-1) + m\left(s\big(B(g_i,k) - B(r_i(t-1),k)\big) + (1-s)\big(B(q_i,k) - B(r_i(t-1),k)\big)\right)$$

Where r_i is the position at dimension i of an initiator in the follower's neighborhood, and q_i is the best location in memory for that initiator. Other variables retain their meaning; essentially, the follower simply copies the change in movement *of the initiator*, by summing this additive factor and its own current velocity. Because the initiator classifies as category 2, its velocity update at iteration t follows the standard RWPSO velocity update equation. If there are two or more initiators in the neighborhood, then the follower follows a baboon behavioral pattern of "compromise" between diverging paths (category 3). This update takes the form of the equation:

$$v_{i,k}(t) = v_{i,k}(t-1) + \frac{\sum_1^z m\left(s\left(B(g_i,k) - B\big(r_{z,i}(t-1),k\big)\right) + (1-s)\left(B\big(q_{z,i},k\big) - B\big(r_{z,i}(t-1),k\big)\right)\right)}{z}$$

Where z represents the number of initiators in the follower's neighborhood, and $r_{z,i}$ and $q_{z,i}$ respectively refer to the current position and best location in memory of neighbor initiator z at dimension i. The velocity update based on this equation will be referred to as "baboon update." The conceptual modification is described as "baboon information-pooling" or "baboon modification."

3.1.2 Constraint Handling over Roulette Wheel Velocities

RWPSO uses a penalty function method to handle infeasibilities arising from unsatisfied constraints. However, if a certain assignment of any dimension within a configuration violates a constraint, then RWPSO simply "unschedules" it, or chooses a neutral assignment that adds no value to the optimization (Smythe, 2012). One of the challenges within the harvest scheduling domain is that feasible solutions may form "islands" essentially surrounded by infeasible solutions. Because RWPSO velocities represent probabilities of selection, assignment of location based on probability still requires a constraint-handling step at the plan formulation stage.

B-RWPSO uses a relaxed constraint-preserving strategy, in which follower particles only traverse feasible positions in the solution space. This is maintained via constraint-checking as solutions are given assignments based on the roulette wheel velocities. In a strict constraint-preserving method, a disconnect may arise between the velocity in the probability space and position in the solution space—for example, a high-probability location *k* for dimension *i* might be rejected for infeasibility in the broader context of the particle's working assignments. This manner of dealing with constraint-preserving is analogous to the harsh repair penalty function from Smythe (2005):

$$f_2 = \sum_{k=1}^{z}\left(T - \sum_{n=1}^{d} V_{n,k}\right)^2$$

$$\text{where } V_{i,t} = \begin{cases} a_n h_{n,k} & if\ s_n = k\ AND\ \nexists q, s_{q\in adj(n)} = k \\ 0 & Otherwise \end{cases} \text{ and}$$

$$adj\ (n) = \{h | h\ is\ a\ stand\ adjacent\ to\ n\}$$

Where unit-restriction adjacency checking is also similar to the simpler graph coloring check employed in the quaternary PSO (Cui & Qin, 2008).

A popular strategy with the constraint-preserving method is the particle "fly-back." This is a default move that forces particles to return to their origin point if they cross into an infeasible region. For continuous search spaces, these boundaries may be more clear, but for nominal, combinatorial problems, constraints may have little relation to spatial locations. Improved constraint-preserving strategies found in the New Vector PSO and Improved Vector PSO alter the fly-back strategy by redirecting a particle entering an infeasible region and updating its velocity (Sun, 2011). There is not a clear analogy between this type of strategy and what could potentially be employed by probability-based discrete variants of the PSO. We implement a modified type of search reversion that uses randomly initialized velocities to overcome the tendency of the algorithm to converge early in local optima that are surrounded by infeasible moves.

3.1.3 Variable RWPSO Parameters

One unique feature of the RWPSO is that it expresses its parameters in explicit terms as coefficients of a roulette wheel-based method. As such, the parameters in the equation are uniquely suited to guide the swarm's traversal over the problem's solution probability space. While current versions of the RWPSO only use static parameters, we introduce conceptual modifications built around the canonical PSO to vary the social/cognitive influence coefficients throughout iterations of the algorithm.

Existing algorithms such as Simulated Annealing and Threshold Accepting employ variable parameters in order to modulate the exploration and exploitation behavior of the algorithm; the modifications to PSO discussed in Chapter 2 (e.g., linear varying inertia weight, time-varying acceleration constants) have similar function. To parallel the usage of time-varying

acceleration coefficients in PSO (Ratnaweera, 2004), we implement a similar strategy (called Time-Varying Parameters for clarity) for the RWPSO's social coefficients. The TVP requires an input s_{start} and s_{finish}, from which the coefficient at time t, $s(t)$ is calculated and linearly increased throughout the algorithm's runtime. The simple calculation is based on:

$$s(t) = s_{start} + \frac{t * \left(s_{finish} - s_{start}\right)}{t_{max}}$$

Because RWPSO and the baboon modified B-RWPSO both use maximum number of iterations as a stopping criteria, calculation of social coefficient at any iteration is based on t_{max}, the total number of allowed iterations.

3.1.4 Reversion

Reversion refers to a technique employed in search algorithms to "kick" a solution from its current location in the search space to a previous recorded best solution. Search reversion has been employed primarily with *s*-metaheuristics. In such cases, reversion interrupts a search sequence by re-initiating the search process from a previously found high-quality or best solution. Reversion techniques often introduce another parameter to the search algorithm, as the reversion rate has been found to have a significant effect on solution quality (Bettinger & Demirci, 2015). In PSO, a particle's immediate position, however, undergoes much more movement than a candidate solution would in other types of search. A type of reversion that has been adapted for the particle swarm by researchers is the concept of velocity "reinitialization". In the context of probability-based PSO, this reinitialization may have a far more significant impact on particle's locations than would a change to particle trajectory in a continuous search space. Following the strategy employed in Ratnaweera (2004) we apply a velocity reinitialization when the velocity (probability) of any location *k* for dimension *i* in a particle becomes 0. If this

happens, the entire dimension *i* for that particle has the velocity for each location *k* in the dimension reset to the reciprocal of the total number of locations. One advantage of the baboon algorithm reinitialization-style reversion is that the reversion technique is implicit in the algorithm, and built into the particle behavior. As such, no parameter testing is needed.

3.1.5 Concept and Implementation

While the essence of the Baboon-Based Roulette Wheel Particle Swarm Optimization is the baboon information-pooling and modified communication procedures drawing from Collective Intelligence, the other modifications are also important as potential changes that could improve the canonical RWPSO. Here, we include a workflow depicting the sequence and structure of the algorithm.

1) Initialize Population, initialize velocities
2) Record particles' best, global best
3) Update particles position based on velocities (this equation is uniform)
4) Call initiators, store their velocity update factors based on standard equation, update initiator velocities
5) If baboon has initiator in neighborhood → becomes follower. Else → update baboon velocity based on standard equation.
6) Update follower baboons' velocities (depending on # of baboons in neighborhood)
7) If termination criteria not reached → loop to step 2. Else → end program

The B-RWPSO was implemented in Python, as was the base RWPSO. Individual baboons (forest plans) were list objects with index corresponding to forest stand number, and the value at a given index in the list corresponding to the time period for harvest of that stand.

3.2 NETWORK INFLUENCE TYPE (LEARNING STRATEGY)

We build a doubly informed B-RWPSO, developing a neighborhood-in-neighborhood approach to define hierarchical levels of social influence. Existing research by Kennedy and Mendes (2006) describes how problem and neighborhood topology can cause differences in the search ability and convergence speed of a fully-informed particle swarm compared to a singly-informed, best-of-neighborhood (canonical) particle swarm. The learning strategy of a particle, or network influence type, thus has important, domain-specific effects on algorithm performance.

3.2.1 Doubly Informed Roulette Wheel Velocity Update Function

We develop a modification of the canonical "best-of-neighborhood" PSO that adds social influence from both the leading particle in the local neighborhood and the leading particle in the whole swarm. This PSO sociometric configuration is referred to as "doubly informed particle swarm." The Fully Informed Particle Swarm (FIPS), which affords social influence from each particle in the swarm on every other, can perform very well on certain types of problems (Mendes & Kennedy, 2004). However, the fully-informed learning strategy is extremal and prone to very poor performance in other scenarios. The Unified Particle Swarm Optimization presented a successful attempt to balance the influence of the local and global leaders in a continuous search space (Parsopolous and Vrahatis, 2005). However, depending on the unification factor used in UPSO to weight *lbest* vs. *gbest*, the equation used by UPSO will not always maintain the roulette wheel nature of velocities in a dimension. Given the parameterization of RWPSO's social and cognitive coefficients, we are able to develop an alternative method of incorporating influence from both a neighborhood and global leader. The doubly-informed learning strategy can be derived by easily adapting the existing roulette wheel

velocity update equation to develop a new function that compromises between the strong social influence of the FIPS and the streamlined best-of-neighbor PSO:

$$v_{i,k}(t) = v_{i,k}(t-1) + m\Big(s\big(B(g_i,k) - B(x_i(t-1),k)\big) + n\big(B(l_i,k) - B(x_i(t-1),k)\big) + (1-s-n)\big(B(p_i,k) - B(x_i(t-1),k)\big)\Big)$$

$$\text{where } B(x_i,k) = \begin{cases} 1 & if\ x_i = k \\ 0 & else \end{cases}$$

The new velocity update function adds two new parameters, n and l_i, which represent the neighborhood influence factor and value at dimension i of neighbor best recorded solution l. While this new partially-informed velocity update equation still incorporates a particle's personal memory and the swarm's leader, it additionally weights the best position found by any particle's leading neighbor. The neighborhoods for this latter social influence are decided in implementation. In order for every particle to be connected to the influence of the global leader, the neighborhood topology is necessarily *gbest*. However, with neighborhood-in-neighborhood hierarchical influence structure, we designate additional social neighborhoods of arbitrary size (assigned by nominal adjacency of particles, as in *lbest*). The employ of influence from social neighbors can improve the balance of exploration and exploitation in the algorithm because social neighbors are distant at runtime, but slowly congregate throughout execution. If an algorithm properly converges, social neighbors become physical neighbors. Thus the neighbor's influence encourages global exploration early on and exploitation later.

In the context of the B-RWPSO, the neighborhood inside the global neighborhood is equivalent to the baboon social neighborhoods. The baboon social neighborhood is defined statically as baboons 1 through $0.05 \times ps$, $0.05 \times (ps + 1)$ through $0.1 \times ps$, and so on where ps

denotes the population size input parameter specified at runtime. The Doubly Informed Particle Swarm network influence type is uniquely compatible with the baboon-based strategy because baboons can be influenced by neighbors chosen both democratically and meritocratically. With the usage of DIPS velocity update, baboons *without* any neighboring movement initiators still receive supplementary social influence from others in their neighborhood.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 TESTING ON EXAMPLE FORESTS

In evaluating heuristics, there are different levels of testing that can be performed to assess the quality of the solutions derived. The preferred method of performance validation in the forest planning domain involves comparison against a mixed integer formulation of the same problem (Bettinger et al., 2009). Alternatively, comparison against a linear programming solution of the relaxed version of the problem can also provide a lower-bound, best-case benchmark for minimization problems. In the case of the 3 test forests, we used a linear programming solution as the "target volume" and harvest goal in the B-RWPSO objective function. For the B-RWPSO, we ran each algorithm 10 times on problems using the 625- or 73-stand forests, and 20 times for problems using the 40-stand forest. We recorded the best solution found by each execution in addition to charting the population's convergence in 20- or 5-iteration timestamps. In order to ascertain "fair" testing compared to the RWPSO, we normalized the number of fitness evaluations allowed for each execution—on the 73-stand and 40-stand forests, this upper bound number of fitness evaluations was 800,000, while the more complicated 625-stand forest was allowed 1,000,000 fitness evaluations. The number of fitness evaluations is a product of population size and epochs/generations of runtime.

In line with findings from the canonical RWPSO, we tested with a static social coefficient of 0.25 as baseline best performance for the RWPSO. With time-varying parameters (TVPs), a wider range of coefficients (including start and end values) were tested. To compare

other algorithmic setups, an identical set of time-varying roulette wheel parameters was implemented across all experiments, due to the superior performance found when testing with TVPs (setting: $m = 0.1$, $s_{start} = 0.1$, $s_{finish} = 0.35$, maxiterations = 5000]). However, these TVPs used for the testing in Table 2 were *not* the optimal parameters, and simply used as a point of reference to compare different experimental conditions. These parameters were held constant to allow for fair comparison between experimental conditions attached to any version of the tested algorithms—RWPSO and B-RWPSO. The grid of experimental settings included:

{Baboon-information pooling: *y / n [B-RWPSO vs. RWPSO]*}

{Velocity-initialization: nonzero-biased start (*pb*) / random-start (*rsv*)}

Tables 2 through 4 show the best and worst solution found across some of these separate executions, the average of their best solutions, and their standard deviation. We compared the results of the B-RWPSO to the best result found by from mixed integer programming in Bettinger & Zhu (2006) (Table 1). We also compared the results of the B-RWPSO to the original RWPSO (Smythe, 2012) and the forestry-specific Raindrop Optimization algorithm (Bettinger & Zhu, 2006).

Test Forest	Objective Function Value	Harvest Volumes			
		Period 1	Period 2	Period 3	Total
625-stand					
Mixed Integer	64,859,941,092	2,796,070	2,834,340	2,851,350	8,661,760
B-RWPSO	71,662,208,452	2,778,050	2,842,570	2,842,100	8,462,720
73-stand					
Mixed Integer	5,500,330	33,049.5	32,933.6	33,399.4	99,382.5
B-RWPSO	5,500,330	33,049.5	32,933.6	33,399.4	99,382.5
40-stand					
Mixed Integer	98,439	8,981.5	8,903.6	8,987.5	26,872.6
B-RWPSO	90,489	8,879.4	9,010.8	9,034.5	26,924.7

Table 1: Best Mixed Integer Solution vs. best B-RWPSO solution (best parameter setting)

The B-RWPSO outperformed mixed integer programming on the simple 73-stand and 40-stand forests. This aligns with expectations and some of the results displayed over the original RWPSO, as PSO is well-known to operate very effectively on small problems. While the B-RWPSO still lags behind the mixed-integer solution for the large 625-stand forest, it only differs by 2.29% in absolute terms of real-world harvested volume. While the more productive solution of mixed-integer programming would be preferred on this problem, these results demonstrate that it is possible for PSO-based algorithms to achieve high-quality (although somewhat less than optimal) results, even on complex and high-dimensional problems. This is still a very useful finding, as heuristics hold the greatest potential value when deployed on complex, high-dimensional problems for which integer programming formulations are difficult or impossible to design. Furthermore, B-RWPSO, like the original RWPSO and other probability-based PSO variants, is conceptually simple to implement because of its nature-inspired mechanisms. The B-RWPSO has an even greater advantage over other PSO variants because the expression of movement over the problem's probability space helps potential users gain clarity into how the algorithm attains solutions. This quality of B-RWPSO should be particularly valued in situations like forest planning.

One other testing consideration was the initialization of velocities (Tables 2-4). The roulette wheel nature of RWPSO demands that the set of probabilities (velocities) across all locations in a dimension must sum to one. Smythe (2012) defaulted the initial velocities $v_{i,k}(t=0)$ to a uniform assignment of the reciprocal of the number of total locations for each location k:

$$[v_{i,0}(0),\ v_{i,1}(0),\ v_{i,2}(0),\ v_{i,3}(0)] = [0.25,\ 0.25,\ 0.25,\ 0.25]$$

This previous literature also suggested that a velocity initialization bias against assignment of the "0" location actually improved convergence speed and solution quality, and found an ideal start

bias of $v_{i,0}(0) = 0.04$ when testing $v_{i,0}(0)$ values along the grid $v_{i,0}(0) \in \{0.01, 0.04, 0.07, 0.10\}$ with the three other assignments given equal probabilities to round out the roulette wheel (Smythe, 2012). We instead randomly initialized velocities by a random roulette wheel generation process.

$$0 \leq v_{i,0}(0), v_{i,1}(0), v_{i,2}(0), v_{i,3}(0) \leq 1$$

$$\text{subject to: } v_{i,0}(0) + v_{i,1}(0) + v_{i,2}(0) + v_{i,3}(0) = 1$$

We found that randomly initialized roulette wheel velocities actually perform better than the original RWPSO's nonzero-bias velocity initialization setting. This contradicts the assumption and previous findings that a forest planning problem would perform better with an initial set of roulette wheel velocities biased against unscheduled, or 0-value, locations. This results in part due to the spatial constraints in the problem; initializing the population with random values allows for more diversification through the search space because throughout the algorithm's execution, fewer moves are restricted by adjacency constraints. This finding sheds light into the interaction between the competing algorithm behaviors of following spatial adjacency constraints and maximizing harvest value. We demonstrate that the evolutionary movement of solutions in SI and other generation-based algorithms interacts with non-native constraint handling strategies, an important consideration for working with algorithms like PSO variants, which lack such an in-built constraint handling mechanism. In harvest scheduling problems, the constraint-preserving method may reduce computational expense by internalizing adjacency checks, but can also limit the exploration capacity of the algorithm. A randomly generated initial population enables the most exploration behavior when coupled with random starting velocities; the global exploration of the algorithm can further be improved with introduction of the baboon information-pooling modification. These novel features for

probability-based PSO help the swarm to converge faster (Figure 6) and to a better final solution (Figures 4,5,6). Besides the value of the baboon communication strategy in identifying the most appropriate *region* for convergence (via enhanced exploration), the baboon modification also improved exploitation within a promising region, ensuring that particles could escape shallow local troughs to find a superior solution in the surrounding area (Figures 4-6).

Alg.	Best (Minimum)	Worst	Average	Std. Dev.
RO	90,499.90	xx	160,698.00	46,879.00
RWPSO (*pb*)	90,489.90	144,672.94	107,518.32	22,730.86
RWPSO (*rsv*)	90,489.90	142,610.82	105,407.40	19,501.98
B-RWPSO (*pb)*	90,489.90	101,863.07	92,636.65	4,345.05
B-RWPSO (*rsv*)	90,489.90	101,863.07	94,280.96	5,686.59

Table 2: Summary statistics for different algorithms applied to the 40-stand Northern Forest

Alg.	Best (Minimum)	Worst	Average	Std. Dev.
RO	5,500,330.28	xx	6,729,995.00	1,472,126.00
RWPSO (*pb*)	5,500,330.28	7,065,589.17	6,015,657.39	628,686.90
RWPSO (*rsv)*	5,560,961.74	6,904,719.75	6,116,777.79	449,938.51
B-RWPSO (*pb*)	5,500,330.28	5,921,566.64	5,584,577.55	188,382.63
B-RWPSO (*rsv*)	5,500,330.28	6,449,654.99	5,606,465.47	297,180.39

Table 3: Summary statistics for different algorithms applied to the 73-stand Western Forest

Alg.	Best (Minimum)	Worst	Average	Std. Dev.
RO	61,913,898,152	xx	66,142,041,314	2,895,384,577
RWPSO (*pb*)	85,439,462,612	113,493,797,732	103,248,427,772	9,370,587,523
RWPSO (*rsv*)	86,452,695,062	94,258,971,212	91,056,808,979	4,087,614,009
B-RWPSO (*pb*)	86,974,967,972	97,131,895,032	89,912,416,257	4,831,133,090
B-RWPSO (*rsv*)	75,481,397,552	81,546,025,732	79,357,171,382	1,588,931,575
B-RWPSO (optimal setting)	71,662,208,452	81,451,900,072	77,703,271,168	3,768,686,890

Table 4: Summary statistics for different algorithms applied to the 625-stand Southern Forest

On the lower-dimensional 40- and 73- stand forests, B-RWPSO runs up against the integer optimum for the problem. While a better "best solution" may not be possible, the B-RWPSO still improves performance compared to RO and the original RWPSO, because B-RWPSO achieves even better average values and a smaller standard deviation. For these simpler

harvest scheduling problems, B-RWPSO achieves a very tight distribution of test values. A small standard deviation in the distribution is especially relevant to the forest planning context because it indicates consistency, reliability, and stability in the heuristic's performance, which allows for increased trust to be placed in the quality of the generated solutions.

4.1.1 Baboon Information-Pooling

To compare the isolated effect of the baboon information-pooling without the confounding effects of other modifications of RWPSO, we run additional tests directly comparing B-RWPSO and RWPSO under otherwise identical experimental settings and parameter setups. We applied these comparison test cases on the high-dimensional 625-stand forest, where the most significant gains can be made to canonical PSO variants. RWPSO, like other types of PSO, struggles on higher-dimensional problems, and is most prone to premature convergence in these complex search spaces. The results from these tests are shown in Tables 5 through 7. Each algorithm was run 10 times for each of the experimental configurations. For each experimental setting, we applied the Shapiro-Wilk normality test on the set of 10 best-found solutions for each of the two algorithms (B-RWPSO and RWPSO) to determine if the sample of best solutions was normally distributed. All of the data listed in Tables 5 through 7 met the normality criteria with $\alpha < .05$. With the assumption of normality in the sampled solution sets, we compared the difference between B-RWPSO and RWPSO for each parameter configuration with a two-tailed *t*-test. For all settings, we observed statistically significant differences ($p < 0.01$) between the RWPSO with baboon information-pooling (B-RWPSO) and standard RWPSO (Tables 5,6,7). The results from these *t*-tests indicate that the distribution of solutions produced in B-RWPSO was significantly different from the distribution of solutions produced by RWPSO.

Algorithm	Best (Minimum)	Worst	Average	Std. Dev.
B-RWPSO	75,481,397,552	81,546,025,732	79,357,171,382	1,588,931,575
RWPSO	86,452,695,062	94,258,971,212	91,056,808,979	4,087,614,009

Table 5: B-RWPSO vs. RWPSO for time-varying parameters $m = 0.1$, $s_{start} = 0.1$, $s_{finish} = 0.35$, *populationsize* = 200; randomly initialized velocity (denoted *rsv*); and *gbest* topology with 10-baboon social neighborhoods. $p = 0.0033$

Algorithm	Best (Minimum)	Worst	Average	Std. Dev.
B-RWPSO (*pb)*	86,974,967,972	99,595,892,492	92,481,591,283	5,093,623,123
RWPSO (*pb)*	85,439,462,612	113,493,797,732	103,248,427,772	9,370,587,523

Table 6: B-RWPSO vs. RWPSO for time-varying parameters $m = 0.1$, $s_{start} = 0.1$, $s_{finish} = 0.35$, *populationsize* = 200; biased-start velocity (denoted *pb)*; and *gbest* topology with 10-baboon social neighborhoods. $p = 0.0041$

Algorithm	Best (Minimum)	Worst	Average	Std. Dev.
B-RWPSO	75,727,943,532	85,773,854,952	80,345,507,369	2,910,402,975
RWPSO	87,444,889,432	108,579,847,892	97,228,699,815	8,933,066,422

Table 7: B-RWPSO vs. RWPSO for time-varying parameters $m = 0.1$, $s_{start} = 0.1$, $s_{finish} = 0.35$, *populationsize* = 200; randomly initialized velocity (denoted *rsv*); *gbest* topology with 10-baboon social neighborhoods; and velocity *reinitialization* "reversion". $p = 0.0047$

Notably, the standard deviation of results from the RWPSO with randomly initialized velocity was smaller than even the standard deviation of the B-RWPSO when B-RWPSO was initialized with the anti-zero starting velocity. Random initialization of velocity dramatically improved performance of both RWPSO and B-RWPSO compared to when each algorithm used the anti-zero velocity bias. On the other hand, the velocity reinitialization (reversion mechanism) actually resulted in inferior performance for both the B-RWPSO and RWPSO. It did not generate substantively different solutions than the B-RWPSO without reinitialization, but had a somewhat inferior standard deviation. However, for the RWPSO, velocity reinitialization significantly decreased performance of the algorithm.

4.1.2 Time-Varying Parameters

The effect of time-varying parameters was also tested. Although there appears to be a slight improvement with time-varying parameters compared to static parameters, it is not a statistically significant difference (Table 8). Likewise, when changing just the s_{finish} value in the experiments, there was very little difference in the performance of the algorithm (Table 9). Although these modifications may not improve performance, this demonstrates that the B-RWPSO is fairly robust to parameterization, which may be useful for deploying the algorithm in a variety of different contexts. The B-RWPSO can be more confidently deployed on a wider variety of problems without extensive parameter testing.

Algorithm	Best (Minimum)	Worst	Average	Std. Dev.
B-RWPSO (TVAC)	86,974,967,972	99,595,892,492	92,481,591,283	5,093,623,123
B-RWPSO (Static)	87,247,858,452	108,539,881,992	94,639,708,121	6,465,199,136

Table 8: B-RWPSO with time-varying parameters [$m = 0.1$, $s_{start} = 0.1$, $s_{finish} = 0.35$] vs. static parameters [$m = 0.05$, $s = 0.25$], *populationsize* = 200; anti-zero biased initial velocity (denoted *pb*); *gbest* topology with 10-baboon social neighborhoods. $p = 0.3077$

Algorithm	Best (Minimum)	Worst	Average	Std. Dev.
B-RWPSO ($s_{\text{finish}} = 0.25$)	75,152,559,312	83,336,276,292	78,998,328,472	3,409,541,240
B-RWPSO ($s_{\text{finish}} = 0.30$)	71,662,208,452	81,451,900,072	77,703,271,168	3,768,686,890
B-RWPSO ($s_{\text{finish}} = 0.35$)	75,727,943,532	85,773,854,952	80,850,313,079	3,216,510,214
B-RWPSO ($s_{\text{finish}} = 0.40$)	74,963,047,112	81,727,568,792	79,611,413,707	3,140,539,631

Table 9: B-RWPSO with time-varying parameters {$m = 0.1$, $s_{start} = 0.05$ and $s_{fin} \in [0.25, 0.3, 0.35, 0.4]$}; *populationsize* = 200; randomly initialized velocity (denoted *bsv*); *gbest* topology with 10-baboon social neighborhoods; velocity *reinitialization* "reversion"

4.1.3 Convergence Behavior

The fitness vs. iteration curves depict the mean best objective found at an iteration for all 10 sample runs of an algorithm at a given parameter setting. On the *y*-axis is the mean calculated

value of the objective function, while the *x*-axis shows number of iterations in various step sizes. (for the 625-stand forest, each tick mark represents 20 iterations, i.e., if the label is 50, we are on the 1000th iteration; for the 40- and 73-stand forests, each tick mark represents 5 iterations.)

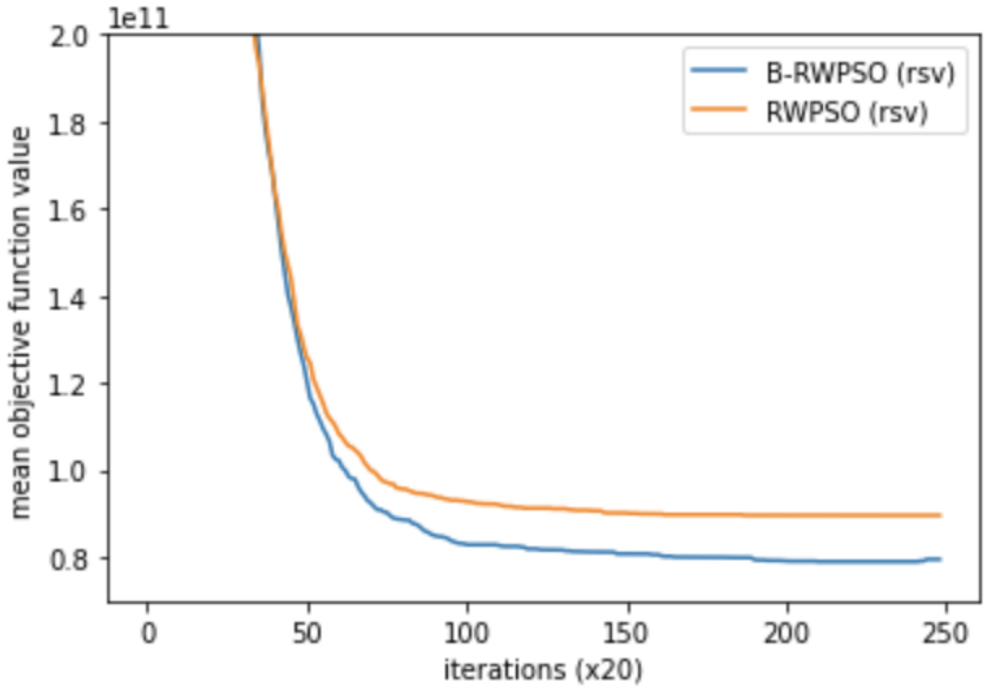


Figure 4: Fitness vs. Iteration curves for B-RWPSO vs. RWPSO on the 625-stand forest, for identical settings besides baboon information-pooling (time-varying parameters $m = 0.1$, $s_{start} = 0.1$, $s_{finish} = 0.35$, *populationsize* = 200; randomly initialized velocity (denoted *rsv*), and *gbest* topology with 10-baboon social neighborhoods.)

For the 625-stand forest, RWPSO and B-RWPSO overlap in the timing of their rapid climb (descent in this case of a minimization problem). At about 1000 iterations, the rate of improvement of solution quality slows down, although B-RWPSO is able to find a superior final solution. For the 73-stand forest, RWPSO actually approaches the local optimum faster than B-RWPSO; however, it suffers premature convergence and plateaus at an inferior final solution compared to B-RWPSO.

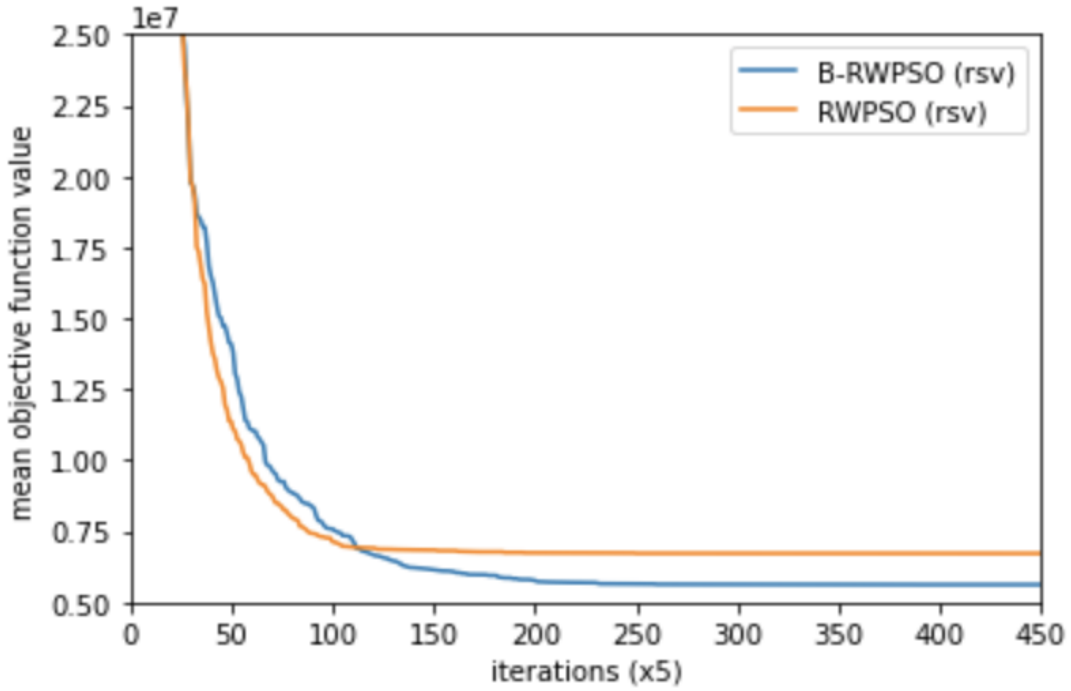


Figure 5: Fitness vs. Iteration curves for B-RWPSO vs. RWPSO on the 73-stand forest, for identical settings besides baboon information-pooling (time-varying parameters $m = 0.1$, $s_{start} = 0.1$, $s_{finish} = 0.35$, *populationsize* = 200; randomly initialized velocity (denoted *rsv*), and *gbest* topology with 10-baboon social neighborhoods.)

Finally, B-RWPSO converges very quickly on the 40-stand forest and with sharp precision. Although B-RWPSO was allowed up to 800,000 or 1,000,000 fitness evaluations for each of these problems, the algorithm finds a solution much quicker than the evaluations allotted. With a population size of 200 for each of these configurations, both B-RWPSO and RWPSO had largely converged by iteration 250, 1000, or 2000 (respectively for the 40-, 73-, and 625-stand forests). These equate to just 5,000, 200,000, and 400,000 fitness evaluations (far less than half the number of evaluations we allowed). We notice that the improvement in performance of the baboon modification over original RWPSO is least significant on this simplest of the test problems.

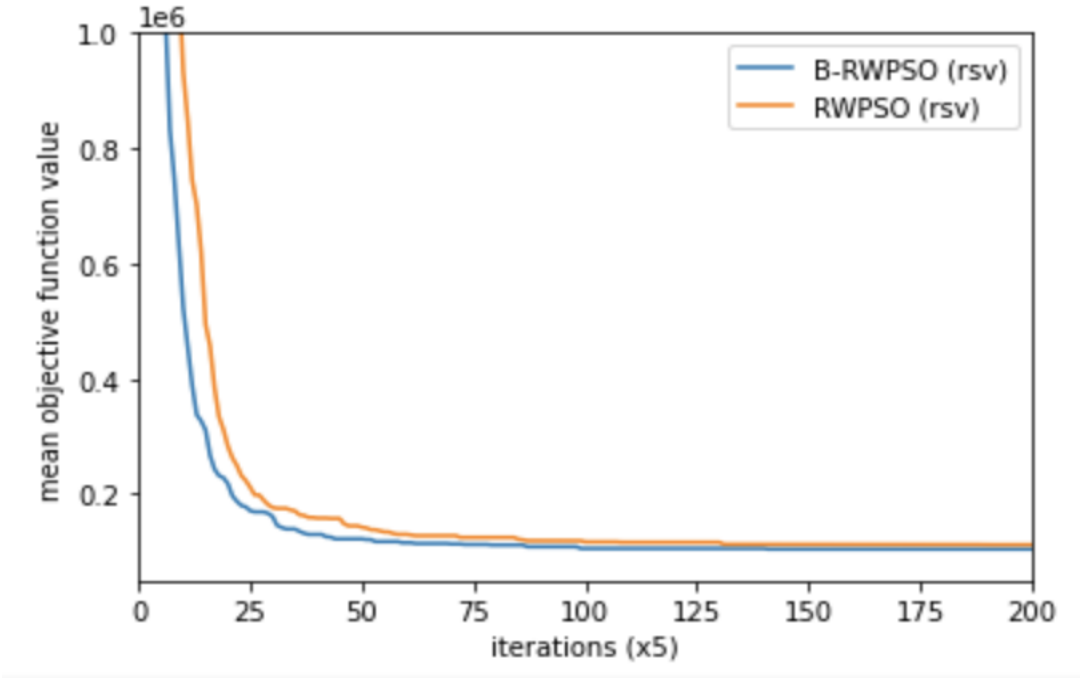


Figure 6: Fitness vs. Iteration curves for B-RWPSO vs. RWPSO on the 40-stand forest, for identical settings besides baboon information-pooling (time-varying parameters $m = 0.1$, $s_{start} = 0.1$, $s_{finish} = 0.35$, *populationsize* = 200; randomly initialized velocity (denoted *rsv*), and *gbest* topology with 10-baboon social neighborhoods.)

These figures depict the exploratory behavior of the algorithms at different *iterations*, in order to establish a uniform point of comparison between progressive stages of the B-RWPSO and RWPSO. However, the two algorithms differed in total runtime, and so B-RWPSO will reach a given iteration more quickly than the canonical RWPSO. Over the runs used in Figure 4 for the 625-stand forest, the baboon modification sped up average algorithm run-time by 22.8%. This advantage is valuable in business applications that favor quick planning.

4.2 NETWORK INFLUENCE TYPE COMPARISON

In FIPS, each particle is informed by the success of all of its neighbors each iteration (Kennedy & Mendes, 2006). Canonical singly informed particle swarm only includes social influence from the global leader in the swarm at each iteration. Doubly informed particle swarm

seeks to balance competing strands of social information by including social influence from both the global leader and a neighborhood leader. We tested these three informational structures on the 625-stand forest:

{Network influence type (static neighborhood) : "best of neighborhood" canonical singly-informed *gbest* / fully-informed *gbest* / doubly-informed *lbest* & *gbest* }

Network Influence Type (Learning Strategy)	Best (Minimum)	Worst	Average	Std. Dev.
Singly informed (*gbest*)	75,481,397,552	81,546,025,732	79,357,171,382	1,588,931,575
Doubly informed (*lbest & gbest*)	79,531,304,832	86,521,861,392	81,941,154,536	2,977,877,438
Fully informed (*gbest*)	77,680,411,052	80,199,673,252	79,777,328,125	1,920,888,834

Table 10: Comparison of Network Influence Types on B-RWPSO with identical parameter settings (time-varying parameters $m = 0.1$, $s_{start} = 0.1$, $s_{finish} = 0.35$, *populationsize* = 200; randomly initialized velocity (denoted *rsv*), and *gbest* topology with 10-baboon social neighborhoods.)

Suitability of a problem to any given network topology and influence type has historically varied by domain. The effects of changing the network's sociometry—whether positive or negative—can be dependent on the topology of the solution space as well as other features of the problem. In the tested cases of harvest scheduling for forest planning, network influence type was not a significant factor in the particle swarm's search behavior. This may be related to the graph-coloring constrained search space, which likely restricts search.

CHAPTER 5

CONCLUSION

We develop a new particle swarm velocity update mechanism based on the communication behavior of wild baboons and apply this modification to a probability-based PSO variant used in harvest scheduling applications for the forest planning problem. Baboon-Based Roulette Wheel Particle Swarm Optimization (B-RWPSO) improves the performance of traditional discrete and probability-based PSO algorithms on a variety of problems, with a particular improvement on complex, high-dimensional harvest scheduling problems. We demonstrate that bio-inspired and nature-inspired strategies may continue to improve existing evolutionary optimization heuristics. Addressing spatial constraints may be difficult in nominal variable problems because boundaries in the solution space are disjoint from real-world geographical boundaries. For probability-based algorithms that traverse over the probability space of the problem, there may be further conflicts with constraint-handling efforts to restrict search to feasible regions.

Our experimental results overturn previous assumptions about harvest scheduling problems that the ideal velocity initialization should be biased *against* unscheduled (or 0-valued) dimensions. Rather, we find that random initialization of velocities (and greater preference toward 0-valued dimensions in early iterations) can actually improve performance of probability-based RWPSO and B-RWPSO. Although this testing only incorporated three harvest scheduling problems, with simplifications compared to real-world planning needs, our results shed light into potential synergies between methods for spatial constraint-handling and algorithm design. These

findings suggest a need for further research into the complex interactions between spatial constraints, optimization objectives, and constraint-handling methods. Additionally, our implementation of a search reversion mechanism via velocity reinitialization resulted in inferior results. Further work may investigate whether this type of reinitialization may be refined, perhaps in a phased implementation, so that it is selectively applied to improve the search.

We also extend research into the role of social influence topology on social interactions in the particle swarm; because the appropriateness of social connectivity is highly dependent on the domain or problem, we test what procedure for information influence will perform best in harvest scheduling with a neighborhood-in-neighborhood approach to include both global and neighbor influence. Finally, while the baboon modification described here was only implemented on the discrete RWPSO, the conceptual baboon information-pooling could hold promising results for canonical and continuous variants of PSO. Among other recent innovations in nature-inspired optimization, the integration of models of collective intelligence may help coordinate information-sharing patterns between sophisticated individuals, especially in complex emerging fields like multi-population swarm intelligence. The baboon modification may also be suitable for Pareto-based multi-objective exploration because subgrouping can enhance exploration of the solution space, and potentially discovery of diverse points along the Pareto frontier.

REFERENCES

Bettinger, P., Demirci, M., and Boston, K. (2015). Search reversion within s-metaheuristics: Impacts illustrated with a forest planning problem. *Silva Fennica*, vol. 49(2): 1-20. https://doi.org/10.14214/sf.1232

Bettinger, P., Sessions, J., and Boston, K. (2009). A review of the status and use of validation procedures for heuristics used in forest planning. *Mathematical and Computational Forestry and Natural-Resource Sciences,* vol. 1(1): 26-37.

Bettinger, P. and Zhu, J. (2006). A new heuristic for solving spatially constrained forest planning problems based on mitigation of infeasibilities radiating outward from a forced choice. *Silva Fennica*, vol. 40(2): 315-333.

Borges, P., Eid, T., and Bergseng, E. (2014). Applying simulated annealing using different methods for the neighborhood search in forest planning problems. *European Journal of Operational Research*, vol. 233(3): 700-710. https://doi.org/10.1016/j.ejor.2013.08.039.

Cui, G., Qin, L., *et al.* (2008). Modified PSO algorithm for solving planar graph coloring problem. *Progress in Natural Science,* vol. 18(3): 353-357.

Eberhart, R., and Kennedy, J. (1995). A new optimizer using particle swarm theory. *Proceedings 6th International Symposium Micro Machine and Human Science*, Nagoya, Japan. 39-43.

Freitas, D., Lopes, L.G., Morgado-Dias, F. (2020). Particle Swarm Optimisation: A historical review up to the current developments. *Entropy*, vol. 22(3): 362. https://doi.org/10.3390/e22030362

Hu, X. and Eberhart, R. (2002). Solving constrained nonlinear optimization problems with particle swarm optimization. *In Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI),* Orlando, FL, USA. vol. 5: 203–206.

Kennedy, J. (2000). Stereotyping: Improving particle swarm performance with cluster analysis. *Proceedings of the Congress on Evolutionary Computation (CEC),* vol. 2: 1507–1512.

Kennedy, J. and Mendes, R. (2006). Neighborhood topologies in fully informed and best-of-neighborhood particle swarms. IEEE Transactions on Systems, Man, and Cybernetics, vol. 36(4): 515-519.

Krause, J., Ruxton, G.D., and Krause, S. (2010). Swarm intelligence in animals and humans. *Trends Ecol Evol.* vol. 25(1): 28-34.

Lawler, E.L. and Wood, D.E. (1966). Branch-and-bound methods: A survey. *Operations Research*, vol. 14: 699-719.

Liu, H., Wang, Y., Tu, L. et al. (2019). A modified particle swarm optimization for large-scale numerical optimizations and engineering design problems. *J Intell Manuf*, vol. 30: 2407–2433. https://doi.org/10.1007/s10845-018-1403-1

Mendes, R., Kennedy, J., and Neves, J. (2004). The fully informed particle swarm: Simpler, maybe better. *IEEE Trans. Evol. Comput.* vol. 8: 204–210.

Parsopolous, K., and Vrahatis, M. (2005). Unified Particle Swarm Optimization for solving constrained engineering optimization problems. *ICNC 2005*, vol. 3612: 582-591

Ratnaweera, A. and Halgamuge, S.K. (2004). Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *IEEE Transactions on Evolutionary Computation*, vol. 8(3): 240–255.

Santos, C.D., Przybyzin, S., Wikelski, M., Dechmann, D.K.N. (2016). Collective decision-making in homing pigeons: Larger flocks take longer to decide but do not make better decisions. *PLoS ONE,* vol. 11(2): e0147497. https://doi.org/10.1371/journal.pone.0147497

Shi, Y. and Eberhart, R.C. (1999). Empirical study of particle swarm optimization. *Proceedings of the Congress on Evolutionary Computation*, vol. 3: 101–106.

Smythe, J., Potter, W., and Bettinger, P. (2012). Application of a new multi-valued Particle Swarm Optimization to forest harvest schedule optimization. *In* Proceedings of the 9th International Conference on Genetic and Evolutionary Methods (GEM'12). Computer Science Research, Education and Applications (CSREA) Press, Las Vegas, NV.

Strandburg-Peshkin, A., Farine, D., et al. (2015). Shared decision-making drives collective movement in wild baboons. *Science*, vol. 348(6241): 1358-1361. https://doi.org/10.1126/science.aaa5099

Sun, C., Zeng, J., and Pan, J. (2011). An improved vector particle swarm optimization for constrained optimization problems. *Information Sciences,* vol. 181(6): 1153-1163. https://doi.org/10.1016/j.ins.2010.11.033

Vasquez, J., Valdez, F., and Melin, P. (2014). Comparative study of social network structures in PSO. *Recent Advances on Hybrid Approaches for Designing Intelligent Systems*. vol. 547: 239-252. https://doi.org/10.1007/978-3-319-05170-3_17