IEEE Swarm Intelligence Symposium (SIS), Honolulu, HI, USA, 2007: 190-197

Source code: http://www.adaptivebox.net/research/fields/framework/miniSwarm/

A Mini-Swarm for the Quadratic Knapsack Problem

Xiao-Feng Xie
Department of Computer Science,
Hong Kong Baptist University,
Kowloon Tong, Hong Kong
Email: xiexf@ieee.org

Jiming Liu
Department of Computer Science,
Hong Kong Baptist University,
Kowloon Tong, Hong Kong
Email: jiming@comp.hkbu.edu.hk

***Abstract*- The 0-1 quadratic knapsack problem (QKP) is a hard computational problem, which is a generalization of the knapsack problem (KP). In this paper, a mini-Swarm system is presented. Each agent, realized with minor declarative knowledge and simple behavioral rules, searches on a structural landscape of the problem through the guided generate-and-test behavior under the law of socially biased individual learning, and cooperates with others by indirect interactions. The formal decomposition of behaviors allows understanding and reusing elemental operators, while utilizes the heuristic information on the landscape. The results on a collection of the QKP instances by mini-Swarm versions are compared with that of both a branch-and-bound algorithm and a greedy genetic algorithm, which show its effectiveness.**

I. INTRO DUCTION

The 0-1 quadratic knapsack problem (QKP) was introduced by Gallo et al. [20], which consists in choosing elements from *n* items for maximizing a quadratic profit objective function subject to a linear capacity constraint. The QKP is a generalization of the 0-1 knapsack problem (KP) [38], by restricting all the quadratic coefficients to zero. The knapsack problems [38] have been intensively studied due to both its theoretical interest and its wide practical applicability. Due to its generality, the QKP has more practically applications [3, 20], moreover, several graph problems can be formulated as the QKP [4, 45].

The KP is NP-hard, although it can be solved exactly in pseudo-polynomial time through dynamic programming based on Bellman recursion [21]. As a generalized version, the QKP is much harder than the KP [10, 45], where the graph structure associated with coefficients of objective function plays a important role in its solution [46].

Due to the NP-hardness in strong sense, the research efforts on solving the QKP may focus on exact methods, approximate algorithms and heuristic methods.

Classical exact methods [39] for solving the QKP are the branch and bound (B&B) algorithms, where numerous upper bounds have been obtained, by using techniques such as derivation of upper planes [20], Lagrangian relaxation [24], reformulation [10], linearization [3], Lagrangian decomposition [4, 5, 41], semidefinite relaxation [26], and reduction strategies [24, 45], etc.

The studies on approximate algorithms, which seek guaranteed polynomial-time performance on approximate solution, were only limited on some special cases of the QKP, such as the classical KP [30], and the QKP where the underlying graph is *edge series-parallel* [46], etc.

The heuristic methods arise from practical interest, which try to find good solution in high probability with acceptable computational efforts by considering certain imprecise knowledge on the structure of the problem. The linearization and exchange (LEX) heuristic [24] uses the best linear L2-approximation to build an associated linear KP and use it for determining items in a greedy way to form a good initial solution, which is then improved by an exchange method between certain items. Hua et al. [29] studied on the convex QKP by an approximate dynamic programming approach. Julstrom [32] also illustrated the capability of several heuristics, including greedy, genetic and greedy genetic algorithms on the QKP.

Swarm algorithms [7, 16, 34] address on tackling a specified optimization task by multiple interacting agents, where each agent works under bounded rationality by executing simple procedural rules according to available heuristic information in iterative cycles. Compared with the individuals in genetic algorithms (GAs) [27], the agents also utilize certain form of socially sharing information for achieving emergent collective behavior, but they work in learning paradigm and are not subjected to the "*survival of the fittest*" principle. Some examples are ant colony optimization (ACO) [15], particle swarm optimization (PSO) [12, 33], and others [25, 54], etc.

The compact multiagent optimization system (MAOS) [56] based on autonomy-oriented computing [37] has been a framework for situating certain algorithms [6, 33, 48]. Similar to the *particle* [33], each agent [56] in the MAOS has a moderate problem solving capability with two basic features: a) addresses on possessing of the long-term declarative memory [1] in limited capacity, which is modified reactively by itself only, instead of operating as sophisticated as in cognitive architectures [1, 35, 50]; and b) achieves complex problem solving by simple guided *generate-and-test* behavior [55, 57] under the law of *socially biased individual learning* (SBIL) [9, 19], a *fast-and-frugal* heuristic [22] under bounded rationality adopted by many species in the real world.

In this paper, a mini-Swarm system for solving the QKP is presented. It is realized in a simplified way from the MAOS [56], while keeps both of its basic features. Specially, each agent searches on a structural landscape of the problem based on extremely limited declarative knowledge, while the socially sharing information is achieved by referring from all agents. Moreover, the formal decomposition of behaviors is addressed explicitly, which allows agents adopting some well-studied operators.

The remainder of the paper is organized as follows. In Section II, the details of the mini-Swarm system are presented. Then in Section III, the QKP itself is introduced. In Section IV, the implementation of a mini-Swarm optimizer for the QKP is described. In Section V, the extensive experimental results by the mini-Swarm, using an online collection of the QKP instances, are compared with those of some existing algorithms [5, 32]. Finally, this paper is concluded in the Section VI.

II. The Mini-Swarm System

The mini-Swarm system consists of a society of N agents, which cooperate in a socially sharing environment (E) [52] for realizing the common intention of finding high-quality solution for an optimization task.

Moreover, the system runs in iterative cycles. For a run, the number of running cycles is T_{RUN}, which is determined by specified termination conditions. At t=0, the system is initialized. The system can be analyzed in each cycle by executing as a Markov chain process, where the system behavior at the tth ($t \in [1, T_{RUN}]_{\mathbb{Z}}$) cycle only depends on the its long-term memorized status at the (t-1)th cycle.

Figure 1 shows one of the agents and the environment it roams. Each agent has a limited long-term declarative memory manipulated by itself only and is communicated with others by indirect interactions, which are implicit implemented through the environment. Moreover, the agents are homogenous in the sense that they have same organization structure, which means all other agents are interacted with the environment in same way.

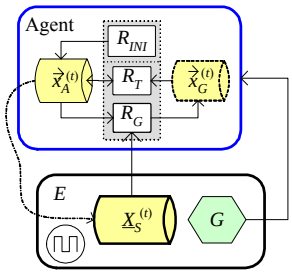

FIGURE 1. AN AGENT AND THE ENVIRONMENT (E) IT ROAMS

A. The environment (E)

The environment (E) serves as the task-dependent domain on which the agents works in cooperative mode, which usually plays three roles [37].

Firstly, it contains a structural *landscape* (G) of the optimization task. The landscape paradigm is used for search in general [8, 28]. It is defined on the *representation space* (S_R), which contains all potential solutions, each is called a *state* ($\vec{x}$). A *quality measuring* operator (R_M) is used for determine which one has better quality between any two states in the *measurable space* (S_M), where $S_M \subseteq S_R$. The quality measurement implies the intention to attain the state that is better than others.

For normal problems, there is a cost function $f_c(\vec{x})$ for each $\vec{x} \in S_M$. Then the quality measurement is realized by $R_M^{\boxed{O}}(\vec{x}_a, \vec{x}_b)$ [56]: for $\forall\ \vec{x}_a, \vec{x}_b \in S_M$, if $f_c(\vec{x}_a) \le f_c(\vec{x}_b)$, then its returns TRUE, which means the quality of the $\vec{x}_a$ is better than that of the $\vec{x}_b$, else it returns FLASE.

Secondly, the environment also acts as a blackboard where agents can share their past local information [42, 55]. As for the *ant* [15], it can be regarded as an indirect communication pool [37]. Here only a single declarative element, i.e. $\underline{X}_S^{(t)} = \{ \vec{x}_{A,i}^{(t)} | i \in [1, N]_{\mathbb{Z}} \}$, is used, where each $\vec{x}_{A,i}^{(t)}$ is referred from the $\vec{x}_A^{(t)}$ of the ith agent.

Thirdly, the environment keeps a central clock that helps synchronizing the behaviors of the system [37]. The system is worked in cycles, where each cycle contains two sequential clock steps: the C_RUN and the C_POST.

B. The agent

Each agent is a socially situated autonomous entity [37], which is able to make decisions for itself, subject to the limitations of the available information.

As a mini cognitive entity, the agent works in cognitive loops [31], where the basic building block is the interplay between memory and behavior [1]. As an elemental solver, the essential behavior of the agent in the loops is guided generate-and-test search [55] on the landscape, as observed in many existing optimization systems [57].

As in Figure 1, basic behaviors include an *initializing* part (R_{INI}), a *generating* part (R_G), and a *testing* part (R_T), which are exerted on two declarative knowledge elements, i.e. the $\vec{x}_A^{(t)}$ and the $\vec{x}_G^{(t)}$, which are memorized by the agent in long-term (be valid in all following cycles) and temporarily (be valid only in current cycle), respectively. Moreover, all the behaviors are driven by an interpret process similar as in a finite-state machine (FSM) [47].

The R_{INI} only works at t=0, which constructs the element $\vec{x}_A^{(0)} \in S_M$ (and hence the corresponding referring element in the $\underline{X}_S^{(0)}$) based on the landscape information.

As t>0, both the R_G and the R_T work on the $\vec{x}_A^{(t)}$ and the $\vec{x}_G^{(t)}$ at the C_RUN step and the C_POST step, respectively. Using two steps allows the R_G of all the agents working under same $\underline{X}_S^{(t)}$ before the $\vec{x}_A^{(t)}$ is changed by the R_T.

The R_G generates a new state $\vec{x}_G^{(t)} \in S_M$ by estimating of distribution for promising space based on two declarative knowledge sources, i.e. the $\vec{x}_A^{(t)}$, which can only be modified by agent itself, and the $\underline{X}_S^{(t)}$, which is socially available from the environment. The law of behavior is socially biased individual learning (SBIL), which is adopted by many species in real world [19]. The SBIL has been suggested to be one of fast-and-frugal heuristics [22], which [9] a) gains most of the advantages of both individual and social learning; and b) facilitates the

emergent properties by allowing learned knowledge to be accumulated from one generation to the next.

The R_T only produces reflex behavior, which replaces the memorized element $\vec{x}_A^{(t)}$ by the generated element $\vec{x}_G^{(t)}$, according a specified criterion. There are two frequent-used R_T versions [56]: a) the *direct* R_T ($R_T^{\boxed{D}}$), which does the action directly; and b) the *better* R_T ($R_T^{\boxed{G}}$), which does the action only as $R_M^{\boxed{O}}(\vec{x}_G^{(t)}, \vec{x}_A^{(t)}) \equiv$ TRUE. It can be seen that the R_T may determine nontrivial properties of the $\vec{x}_A^{(t)}$: the $\vec{x}_A^{(t)}$ stores the most recently state or the best state ever obtained as the $R_T^{\boxed{D}}$ or the $R_T^{\boxed{G}}$ is used, respectively.

C. The decomposition of behaviors

All the behaviors may be realized in various ways, and some of them may be sophisticated. For the purpose of understanding, reusing, and even evolving them, it is preferred to interpret them by some elemental operators organized by symbolic scripts. The possible organizing forms and related operators may be numerously. However, only simple variants may be considered. Moreover, it will be especially beneficial to include well-studied operators, by considering the inherent information of each behavior.

Both the R_{INI} and the R_G generate a state in the S_M. For certain landscape, its S_M is not equal to the S_R. Hence it is useful to include a postprocessing step, i.e. a *state-refining* operator (R_{SR}) for refining $\vec{x} \in S_R$ into $\vec{x} \in S_M$.

Then the early process step of both behaviors can focus on generating a state $\vec{x} \in S_R$. For the R_{INI}, this step is realized by an *internal state-constructing* operator (R_{SCI}). A simple version for the R_{SCI} is a *random* R_{SCI} ($R_{SCI}^{\boxed{R}}$), which constructs a state $\vec{x}$ in the S_R at random.

The R_G has two additional input sources, i.e. the $\vec{x}_A^{(t)}$ and the $\underline{X}_S^{(t)}$, which is often filtered into an intermediate form (V_{GF}) by a preprocessing step, i.e. an *information-filtering* operator (R_{IF}), since certain information may be useless. Hence, its early process step contains two sequential steps, i.e. a R_{IF} operator and an *internal state-generating* operator (R_{SGI}), which generates a state $\vec{x} \in S_R$ according to the V_{GF}. Obviously, both the R_{IF} and the R_{SGI} could only be designed under a specified V_{GF}.

For the V_{GF} of the R_G under the law of SBIL, the $\vec{x}_A^{(t)}$ is always kept, normal as an incumbent state for *kick-move*, and the filtering operation is only performed on the $\underline{X}_S^{(t)}$.

The R_{IF} operator may be degenerated as some R_{SGI} operators, such as the social impact operator [6, 56] for numerical optimization, and Inver-over operator [51] for the TSP, etc., may use full information in the $\underline{X}_S^{(t)}$.

Some R_{SGI} operators may use partial information. For example, in the differential evolution (DE) [48, 56], only the best one and pairs of states in the $\underline{X}_S^{(t)}$ are used.

Especially, for $V_{GF}^{\boxed{X}} = \{ \vec{x}_{P1}^{(t)}, \vec{x}_{P2}^{(t)} \}$ where $\vec{x}_{P1}^{(t)} = \vec{x}_A^{(t)}$ and $\vec{x}_{P2}^{(t)}$ is picked from $\underline{X}_S^{(t)}$ by a state-picking operator (R_{SP}), then the R_{SGI} is called as a *recombination* one ($R_{SGI:X}$). By taking both $\vec{x}_{P1}^{(t)}$ and $\vec{x}_{P2}^{(t)}$ as parents, many recombination or crossover operators, which have been well studied in GAs, can be taken as $R_{SGI:X}$ with minor modifications. For this case, the tuple $<R_{IF}, R_{SGI}>$ can be represented by the tuple $<R_{SP}, R_{SGI:X}>$. A simple version of the R_{SP}, called as the $R_{SP}^{\boxed{R}}$, is to pick a state from the $\underline{X}_S^{(t)}$ at random.

D. Relations with existing systems

The mini-Swarm system can be situated in the compact MAOS [56] with almost minimum requirement on declarative knowledge. Similar to that of self-organizing particle systems [47] in artificial life society, each agent works by organized behaviors on limited private memory.

With suitable landscape, several existing algorithms [6, 48, 51] can be well situated in the mini-Swarm framework.

For the socially sharing information, it does not utilize sophisticated external memory, as pheromone trials for the *ant* [15]; for the declarative knowledge of each agent, it does not use multiple elements, as in the *particle* [33].

In principle, the R_G may also work under the *individual-only* mode or the *social-only* mode, which only uses the $\vec{x}_A^{(t)}$ or the $\underline{X}_S^{(t)}$, respectively. For the former mode, the mini-Swarm turns into N independent solvers in parallel running mode, as a *portfolio of algorithms* [23]. For the latter mode, the mini-Swarm is almost equivalent to a *generational* genetic algorithm [53] as two conditions are satisfied: a) for the R_G, a $R_{SGI:X}$ operator is used under $V_{GF}^{\boxed{X}}$, where both the $\vec{x}_{P1}^{(t)}$ and the $\vec{x}_{P2}^{(t)}$ are selected from the $\underline{X}_S^{(t)}$; and b) for the R_T, the $R_T^{\boxed{D}}$ is used.

III. The Quadratic Knapsack Problem (QKP)

The QKP may be expressed as follows [3]:

$$\begin{aligned} &\max\ f(\vec{x}) = \sum_{i=1}^{n} p_i x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} p_{ij} x_i x_j \\ &\text{subject to}\quad g(\vec{x}) = \sum_{i=1}^{n} w_i x_i - c \le 0 \\ &\vec{x} = (x_1, \ldots, x_n)\quad x_i \in \{0,1\}\quad i \in [1,n]_{\mathbb{N}} \end{aligned} \tag{1}$$

where w_i is weight coefficient, p_i and p_{ij} are profit coefficients, and c denotes the capacity of the knapsack. All coefficients are non-negative integers. In order to avoid trivial problem, it is assumed that $c/w_{max}<1$, where $w_{max}=\sum_{i=1}^{n} w_i$. The $g(\vec{x})$ is also called as the *capacity gap*. For each $\vec{x}$, if it satisfies the capacity constraint $g(\vec{x}) \le 0$, then it is called as *feasible*, or else it is called as *infeasible*.

As x_i =1, it is said that the item i is selected. Hence, for each $\vec{x}$, there is an equivalent representation called as $\underline{x}$, which is the set of indices for all the selected items.

The number of non-zero coefficients of the objective function divided by $n\cdot(n+1)/2$ is called as *density* (d) [3], which indicates the level of interaction between items and may have influence on the problem difficulty.

IV. Implementations for the QKP

The implantation for the QKP is realized in the following steps: a) defines the internal representation of the QKP; b) designs suitable elemental operators for the R_{INI}, the R_G, and the R_T behaviors; and c) organizes available operators into versions of the mini-Swarm.

A. The QKP landscape

The n-dimensional binary space for containing possible $\vec{x}$ is a natural choice for the S_R of the QKP. However, the quality measurement is only applied on the *feasible space* (S_F), i.e. the set of all feasible states, which allows a simple cost function, i.e., $f_c(\vec{x}) = -f(\vec{x})$.

B. The elemental operators

Here only two elemental operators are described, where other operators that are not associated with specific problem structure are introduced in previous sections.

1).The recombination R_{SGI} operator

Since each one is a bitstring for both parents, the uniform crossover (UX) [49] is used as the $R_{SGI:X}$ operator, which is called as $R_{SGI:X}^{\boxed{UX}}$ at here. It begins by including in the offspring state all common items in both parent states, then the remained items in both parents are then picked with an equal probability. Compared with the intersection operator as used in a greedy GA [32], which only includes all common items, the UX preserves more information contained in the parents, although it cannot always generate a feasible offspring even as both parents are feasible. Compared with the one-point and the two-point crossovers, the state generated by UX has less variance on the number of selected items related to its parent states. Moreover, it is straightforward to define the *distance* between two states as the total number of items different from each others. The UX also generates an offspring state with similar distance to both parent states, as in the distance preserving crossover (DPX) [18], which may be especially useful for escaping from local minima.

The UX has been applied on solving multidimensional knapsack problem (MKP) by GA [11]. The difference on using it is that here it is no combined with an immediately mutation operator in a small mutation rate.

2).The repairing & improving R_{SR} operator

A feature of the QKP landscape is that $S_M \equiv S_F$. The *repairing & improving* R_{SR} operator ($R_{SR:RI}$) handling such fact by combing two sequential executed operators [11]: a) the *repairing* phase (R_{REP}), which is to repair an state into a feasible one; and b) the *improving* phase (R_{IMP}), which tries to improve the quality of a feasible state.

Concerned with the QKP, for $\forall\vec{x}_a$ has $g(\vec{x}_a)>0$, it can be rather easily to repair the $\vec{x}_a$ into a state $\vec{x}_b \in S_F$, where $\underline{x}_b \subset \underline{x}_a$, by removing some items in the $\underline{x}_a$.

For the repairing phase, a *random* R_{REP} operator ($R_{REP}^{\boxed{R}}$) is realized: for $\forall\vec{x} \in S_R$, each item picked in $\underline{x}$ at random is iterative removed until there is $g(\vec{x}) \le 0$.

The candidate set $\underline{x}_c$ for $\forall\vec{x} \in S_F$ is a full collection of items, where x_i is not selected and satisfies $w_i + g(\vec{x}) \le 0$. If $\underline{x}_c \equiv \varnothing$, then the $\vec{x}$ is called as a *complete*, or else it is called as a *incomplete*. Each incomplete feasible state $\vec{x}$ can be further improved for its $f_c(\vec{x})$ while keeping its feasibility by adding any one of the items in its $\underline{x}_c$. Hence the improving phase can be achieved by a construction process, called $R_{IMP:C}$, for $\forall\vec{x} \in S_F$. At each iteration, the $\underline{x}_c$ is updated, and an item picked in the $\underline{x}_c$ is added into the $\vec{x}$, this process is continued until $\underline{x}_c \equiv \varnothing$.

The updating of $\underline{x}_c$ will be executed many times, which can be accelerated by establishing an intermediate array in advance with the indices of all items sorted by their w_i.

For the $R_{IMP:C}$, a cost-based item-picking version is considered. Similar to the construction phase of the GRASP [17], each item in the $\underline{x}_c$ has an *incremental cost* (C_{INC}) assigned by a *cost-assigning* operator (R_{CA}), which is associated with clues from the problem. Then the *item-picking* operator (R_{IP}) picks a item according the cost values of all the items in the $\underline{x}_c$, where a larger cost value means higher probability for the item to be picked.

For knapsack problems, the profit-to-weight ratio (r_{pw}) associated with each candidate item provides strong correlated information. In his pioneer work, Danzig [14] provided an elegant way of finding an exact solution for the continuous 0-1 Knapsack version by sorting the items according to their r_{pw} values. Balas [2] then defined the *core problem* for the 0-1 Knapsack problem, which may focus the computational effort on a small subset of the items, i.e. the *core*. Of course, only the approximate core, which may be specified in relative exactly, can be used since the optimal solution should be known in advance for determining the exact core. However, the core problems are still generally difficult to be solved [43].

For the QKP, the *absolute profit* (P_A) for the ith item is:

$$P_{A,i} = p_i + \sum_{j \ne i} \tilde{p}_{ij} \tag{2}$$

where $\tilde{p}_{ij} = p_{ij}$ for $i<j$, and $\tilde{p}_{ij} = p_{ji}$ for $i>j$.

The *relative profit* (P_R) for the ith item of a state $\vec{x}$ is:

$$P_{R,i} = p_i + \sum_{j \in \underline{x}} \tilde{p}_{ij} \tag{3}$$

The C_{INC} value for each item must have positive correlation with the probability of the item belonged to a high-quality solution. Hence for knapsack problems, it is

naturally for assigning the r_{pw} as the C_{INC} for each item, as a fast-and-frugal clue could be readily available.

Either absolute profit (P_A) or relative profit (P_R) can be used for calculating the r_{pw} values [32]. The P_R has richer information than the P_A, however, the computation is more costly since it must be re-calculated for each $\vec{x}$.

Here the improving phase is achieved by the $R_{IMP:C}^{\boxed{TS\&A}}$, which can be represented by a tuple $< R_{CA}^{\boxed{A}}, R_{IP}^{\boxed{TS}} >$.

The *absolute* R_{CA} ($R_{CA}^{\boxed{A}}$) is realized as: for each item i, $r_{pw,i} = P_{A,i}/w_i$ is returned as its C_{INC} value.

The *tournament-selection* R_{IP} ($R_{IP}^{\boxed{TS}}$) returns the item with the largest C_{INC} value among totally MIN(N_{TS}, N_{CXC}) items chosen in the $\underline{x}_c$ at random, where N_{TS} is tournament size and N_{CXC} is the cardinality of the $\underline{x}_c$. If N_{TS}=1, then the item is selected at random. If $N_{TS} \geq N_{CXC}$, then the item with minimal C_{INC} value in the $\underline{x}_c$ is selected.

C. The mini-Swarm optimizer

Now an optimizer under the mini-Swarm system may be established with specified elemental operators.

For behaviors, the $R_T^{\boxed{G}}$ is adopted as the R_T. The R_{INI} is realized by $R_{SCI}^{\boxed{R}}$ and $R_{SR:RI}^{\boxed{R\&TS}}$, and the R_G is organized by a tuple $< R_{SP}^{\boxed{R}}, R_{SGI:X}^{\boxed{UX}}, R_{SR:RI}^{\boxed{R\&TS}} >$ under the $V_{GF}^{\boxed{X}}$. Here $R_{SR:RI}^{\boxed{R\&TS}}$ $=< R_{REP}^{\boxed{R}}, R_{IMP:C}^{\boxed{TS\&A}} >$ is reused by both the R_{INI} and the R_G.

Then the algorithm parameters only include the number of agent (N) and the N_{TS} for the $R_{IP}^{\boxed{TS}}$ of the $R_{IMP:C}^{\boxed{TS\&A}}$.

V. Results and Discussions

The experiments were performed on a collection of QKP instances provided by Billionnet and Soutif [5] (URL: http://cedric.cnam.fr/~soutif/QKP/). There are two choices for the number of items (n), i.e. 100 and 200; and four choices for density (d), i.e. 0.25, 0.50, 0.75, and 1.00. For each specified n and d, there are ten instances. For the profit coefficients (p_i and p_{ij}), zero elements are specified at random, and the non-zero elements are uniformly distributed between $[1, 100]_{\mathbb{Z}}$, the weight coefficients (w_i) are chosen between $[1, 50]_{\mathbb{Z}}$ at random, and the capacity (c) is randomly selected between $[50, \max(50, w_{max})]_{\mathbb{Z}}$.

We have the following mini-swarm versions: a) #STD, where N_{TS}=2; b) #RND, where N_{TS}=1; and c) #GRD, where N_{TS}=n. These versions may run under specified N values. For the termination condition, the number of maximum cycles (T_{MAX}) is fixed as 500, and the system is terminated if no further improvement occurs for 100 cycles. The mini-Swarm is coded in JAVA, and run by Sun JVM version 1.5 on the 3.06GHz P4 processor under Red Hat Linux 7.3. For each problem instance, 100 independent trials (N_R) were performed.

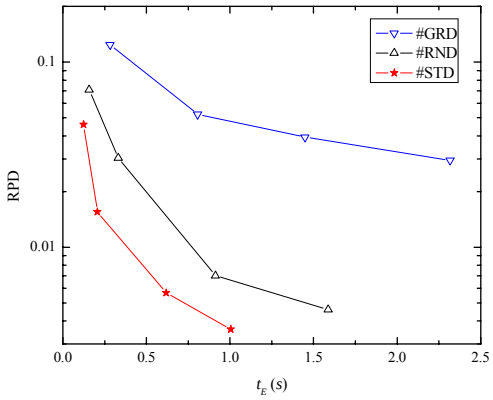

FIGURE 2. RESULTS BY THREE MINI-SWARM VERSIONS

TABLE 1. RESULTS BY #STD FOR 100-ITEM QKP INSTANCES

Instance	F^*	N_S	RPD	T_{RUN}	t_E (s)
r_100_025_01	18558	100	0	56.12	0.430
r_100_025_02	56525	100	0	12.86	0.170
r_100_025_03	3752	100	0	24.89	0.190
r_100_025_04	50382	100	0	25.12	0.920
r_100_025_05	61494	100	0	0.06	0.064
r_100_025_06	36360	100	0	45.24	0.415
r_100_025_07	14657	100	0	27.36	0.233
r_100_025_08	20452	100	0	27.06	0.313
r_100_025_09	35438	39	0.117	58.07	0.801
r_100_025_10	24930	100	0	45.10	0.890
r_100_050_01	83742	100	0	24.93	0.273
r_100_050_02	104856	54	0.028	19.69	0.224
r_100_050_03	34006	88	0.008	48.47	0.366
r_100_050_04	105996	100	0	8.82	0.137
r_100_050_05	56464	100	0	30.06	0.669
r_100_050_06	16083	100	0	30.92	0.263
r_100_050_07	52819	100	0	36.36	0.417
r_100_050_08	54246	100	0	37.69	0.416
r_100_050_09	68974	100	0	34.99	0.950
r_100_050_10	88634	100	0	35.48	0.345
r_100_075_01	189137	100	0	0	0.090
r_100_075_02	95074	75	0.014	72.32	0.749
r_100_075_03	62098	100	0	30.89	0.349
r_100_075_04	72245	100	0	42.40	0.522
r_100_075_05	27616	100	0	23.23	0.210
r_100_075_06	145273	100	0	13.64	0.555
r_100_075_07	110979	100	0	34.52	0.341
r_100_075_08	19570	100	0	25.67	0.192
r_100_075_09	104341	100	0	33.81	0.417
r_100_075_10	143740	100	0	17.79	0.209
r_100_100_01	81978	100	0	28.63	0.271
r_100_100_02	190424	100	0	20.59	0.230
r_100_100_03	225434	100	0	2.41	0.190
r_100_100_04	63028	100	0	32.70	0.400
r_100_100_05	230076	100	0	3.23	0.106
r_100_100_06	74358	100	0	29.66	0.330
r_100_100_07	10330	100	0	11.23	0.120
r_100_100_08	62582	100	0	27.49	0.257
r_100_100_09	232754	100	0	5.55	0.109
r_100_100_10	193262	100	0	18.10	0.240

TABLE 2. RESULTS BY #STD FOR 200-ITEM QKP INSTANCES

Instance	F^*	N_S	RPD	T_{RUN}	t_E (s)
r_200_025_01	204441	100	0	44.59	1.221
r_200_025_02	239573	100	0	14.94	0.499
r_200_025_03	245463	98	6.8E-4	14.72	0.536
r_200_025_04	222361	100	0	23.69	0.726
r_200_025_05	187324	100	0	65.64	1.571
r_200_025_06	80351	28	0.054	136.89	2.781
r_200_025_07	59036	77	0.036	136.10	2.583
r_200_025_08	149433	100	0	99.37	2.038
r_200_025_09	49366	100	0	87.78	1.405
r_200_025_10	48459	100	0	67.73	0.940
r_200_050_01	372097	100	0	45.57	1.292
r_200_050_02	211130	26	0.005	106.65	1.825
r_200_050_03	227185	100	0	88.98	1.606
r_200_050_04	228572	100	0	62.89	4.801
r_200_050_05	479651	100	0	12.12	0.501
r_200_050_06	426777	100	0	39.34	1.053
r_200_050_07	220890	100	0	93.90	2.066
r_200_050_08	317952	100	0	72.20	1.726
r_200_050_09	104936	100	0	77.22	1.097
r_200_050_10	284751	98	4.2E-5	94.87	2.079
r_200_075_01	442894	16	0.025	129.64	3.339
r_200_075_02	286643	95	5.2E-5	129.06	2.797
r_200_075_03	61924	100	0	42.74	0.656
r_200_075_04	128351	100	0	63.19	1.232
r_200_075_05	137885	100	0	62.03	1.127
r_200_075_06	229631	100	0	79.66	1.661
r_200_075_07	269887	100	0	86.67	6.291
r_200_075_08	600858	98	1.5E-4	82.42	1.966
r_200_075_09	516771	100	0	64.34	1.515
r_200_075_10	142694	100	0	60.94	1.072
r_200_100_01	937149	100	0	16.62	0.574
r_200_100_02	303058	100	0	64.75	1.179
r_200_100_03	29367	100	0	27.43	0.301
r_200_100_04	100838	100	0	43.22	0.543
r_200_100_05	786635	100	0	66.13	2.043
r_200_100_06	41171	100	0	25.49	0.311
r_200_100_07	701094	100	0	77.89	1.732
r_200_100_08	782443	100	0	49.84	1.258
r_200_100_09	628992	100	0	105.31	2.277
r_200_100_10	378442	100	0	82.26	1.757

Figure 2 summarizes the average results of all the QKP instances for their relative percentage deviation (RPD) versus the average execution time (t_E) in seconds, by #STD, #RND, and #GRD under N=50, 100, 300, and 500. It can be seems that the #STD can achieve dominant results by comparing to that of both the #RND and the #GRD, and the #RND can achieve dominant results by comparing to that of the #GRD. Hence, the profit-to-weight ratio values indeed carry positive clue, however, to use them in greedy way is normally not a good choice.

Table 1 and 2 lists the summary of the optimum values (F^*) and the performance indices, including the number of successful trials for achieving the optimum (N_S), the RPD, the average running cycles (T_{RUN}), and the t_E (in seconds), by the #STD under N=500 for all 100- and 200-item QKP instances, respectively. It can be seen that about 85% of the instances can be solved in 100% success ratio (N_S/N_R).

Table 3 compares the average time (in seconds) for solving exactly by a depth-first-search branch-and-bound algorithm (B&B) [5] with an relatively original variable fixing procedure and both the average execution time (in seconds) and the success ratio (N_S/N_R) of the #STD mini-Swarm version under N=500, which can be summarized from Table 1 and 2, for the QKP instances under different n and d. The B&B algorithm has outperformed the performance two other algorithms [10, 24] that are able to handle the QKP instances of 100 items or more. This B&B algorithm was implemented in C language and the tests were run on a 300MHz Pentium II processor. For the B&B algorithm, it may be failed for solving the 200-item instances with large density values. For the mini-Swarm version, it can solve with much less computation time than the B&B while always obtain the success ratio larger than 90% for all the instances in different n and d. Moreover, the mini-Swarm version is especially efficient for solving the instances with d=1.00, which may due to a larger variation between the profit and weight coefficients for the knapsack problems with larger density [44].

TABLE 3. RESULTS BY B&B [5] AND #STD UNDER N=500

	n=100		n=200	
d	B&B	#STD(N=500)	B&B	#STD(N=500)
0.25	117	0.442(93.9%)	3602	1.430(90.3%)
0.50	82	0.406(94.2%)	1690	1.805(92.4%)
0.75	120	0.363(97.5%)	-	2.165(90.9%)
1.00	190	0.225(100%)	-	1.197(100%)

"-" means that at least one instance could not be solved within 30000 seconds [5].

TABLE 4. RESULTS BY GGA [32] AND #STD UNDER DIFFERENT N VALUES FOR QKP INSTANCES (n=100, d=0.25)

	GGA	N=100	N=300	N=500
RPD	0.034	0.035	0.016	0.012
N_S/N_R	89.2%	84.1%	91.3%	93.9%
t_E (s)	1.144	0.090	0.310	0.442

TABLE 5. RESULTS BY GGA [32] AND #STD UNDER DIFFERENT N VALUES FOR QKP INSTANCES (n=200, d=1.00)

	GGA	N=100	N=300	N=500
RPD	0.001	0.014	2.2E-5	0
N_S/N_R	91.2%	92.3%	99.9%	100%
t_E (s)	15.78	0.302	0.716	1.197

Table 4 and 5 summaries three performance indices, including the RPD, the success ratio (N_S/N_R), and the t_E (in seconds), for the results by the greedy GA (GGA) [32], and the #STD under N=100, 300, and 500, on the QKP instances as (n=100, d=0.25) and (n=200, d=1.00), respectively. The GGA has achieved best performance while comparing to two greedy heuristics and a naive GA [32]. Moreover, it is generational [53] and 1-elitist, while applies crossover and mutation independently in fixed probability on tournament selected individuals. It was implemented in C++ and executed on the 2.53GHz P4 processor under the Red Hat Linux 9.0. Due to the

restriction on information diffusion by the private memory of each agent, the mini-Swarm has more capability for exploring in parallel, and it is not so necessarily to maintain the diversity of socially sharing information by explicit mutations. As N=100, the #STD version can obtain comparable performance with the GGA, while the #STD versions under both N=300 and N=500 can achieve dominant results on all three performance indices by considering the minor difference between the CPU speed.

VI. Conclusions

The QKP is a hard computational problem with a variety of applications. Due to its NP-hardness in strong sense, the studies on heuristic methods arise from practical interest, especially for the large sized QKP.

Swarm algorithms may be considered as *intelligent* heuristic methods, which can be applied to a variety of optimization problems. In the mini-Swarm framework, each agent searches on the landscape according to guided generate-and-test behavior under the law of socially-biased individual learning, which is based on extremely limited declarative knowledge and interacts with others through the socially sharing knowledge referred from all agents. The formal decomposition of behavioral rules allows the agents utilizing well-studied operators. The relations with some existing methods are then discussed.

The extensive experimental results by the mini-Swarm versions on a collection of the QKP instances show its efficiency as comparing to some existing algorithms.

Further studies may focus on applying the mini-Swarm on various optimization problems, including dynamic [36], multi-objective [13], or multi-constraint [11] cases, etc., and investigating various operators suitable for those problems, which may also be used by some autonomous agents and multiagent systems. Moreover, it is also interesting to study characteristics of swarm algorithms under various network topologies [40].

References

[1]J. R. Anderson, *Learning and Memory: an Integrated Approach*. New York: Wiley, 1995.

[2]E. Balas and E. Zemel, "An algorithm for large zero-one knapsack problems," *Operations Research*, vol. 28, pp. 1130-1154, 1980.

[3]A. Billionnet and F. Calmels, "Linear programming for the 0-1 quadratic knapsack problem," *European Journal of Operational Research*, vol. 92, pp. 310-325, 1996.

[4]A. Billionnet, A. Faye, and É. Soutif, "A new upper bound for the 0-1 quadratic knapsack problem," *European Journal of Operational Research*, vol. 112, pp. 664-672, 1999.

[5]A. Billionnet and É. Soutif, "An exact method based on Lagrangian decomposition for the 0-1 quadratic knapsack problem," *European Journal of Operational Research*, vol. 157, pp. 565-575, 2004.

[6]S. I. Birbil, S.-C. Fang, and R.-L. Sheu, "On the convergence of a population-based global optimization algorithm," *Journal of Global Optimization*, vol. 30, pp. 301-318, 2004.

[7]E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*. UK: Oxford University Press, 1999.

[8]Y. Borenstein and R. Poli, "Information landscapes," *Genetic and Evolutionary Computation Conference*, Washington DC, USA, pp. 1515-1522, 2005.

[9]R. Boyd and P. J. Richerson, *The Origin and Evolution of Cultures*. New York: Oxford University Press, 2005.

[10] A. Caprara, D. Pisinger, and P. Toth, "Exact solution of the Quadratic Knapsack Problem," *INFORMS Journal on Computing*, vol. 11, pp. 125-137, 1999.

[11] P. C. Chu and J. E. Beasley, "A genetic algorithm for the multidimensional knapsack problem," *Journal of Heuristics*, vol. 4, pp. 63-86, 1998.

[12] M. Clerc, *Particle Swarm Optimization*. London: ISTE Publishing Company, 2006.

[13] C. A. C. Coello, G. T. Pulido, and M. S. Lechuga, "Handling multiple objectives with particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, pp. 256-279, 2004.

[14] G. B. Dantzig, "Discrete-variable extremum problems," *Operations Research*, vol. 5, pp. 266-277, 1957.

[15] M. Dorigo, V. Maniezzo, and A. Colorni, "The ant system: optimization by a colony of cooperating agents," *IEEE Trans. Systems, Man, and Cybernetics - Part B*, vol. 26, pp. 1-13, 1996.

[16] A. P. Engelbrecht, *Fundamentals of Computational Swarm Intelligence*: Wiley & Sons, 2005.

[17] T. A. Feo and M. G. C. Resende, "Greedy randomized adaptive search procedures," *Journal of Global Optimization*, vol. 6, pp. 109-133, 1995.

[18] B. Freisleben and P. Merz, "New genetic local search operators for the traveling salesman problem," *International Conference on Parallel Problem Solving from Nature*, Berlin, Germany, pp. 890 - 899, 1996.

[19] B. G. Galef and K. N. Laland, "Social learning in animals: Empirical studies and theoretical models," *Bioscience*, vol. 55, pp. 489-499, 2005.

[20] G. Gallo, P. L. Hammer, and B. Simeone, "Quadratic knapsack problems," *Mathematical Programming Study*, vol. 12, pp. 132-149, 1980.

[21] M. R. Garey and D. S. Johnson, ""Strong" NP-completeness results: motivation, examples, and implications," *Journal of the ACM*, vol. 25, pp. 499-508, 1978.

[22] G. Gigerenzer and D. G. Goldstein, "Reasoning the fast and frugal way: models of bounded rationality," *Psychological Review*, vol. 103, pp. 650-669, 1996.

[23] C. P. Gomes and B. Selman, "Algorithm portfolio design: theory vs. practice," *Conference On Uncertainty in Artificial Intelligence*, New Providence, RI, pp. 190-197, 1997.

[24] P. L. Hammer and D. J. Rader, "Efficient methods for solving quadratic 0-1 knapsack problems," *INFOR*, vol. 35, pp. 170-182, 1997.

[25] S. He, Q. H. Wu, and J. R. Saunders, "A novel group search optimizer inspired by animal behavioural ecology," *IEEE Congress on Evolutionary Computation*, Vancouver, BC, Canada, pp. 1272-1278, 2006.

[26] C. Helmberg, F. Rendl, and R. Weismantel, "A semidefinite programming approach to the quadratic knapsack problem," *Journal of Combinatorial Optimization*, vol. 4, pp. 197-215, 2000.

[27] J. H. Holland, *Adaptation in natural and artificial systems*. Ann Arbor: University of Michigan Press, 1975.

[28] W. Hordijk, "A measure of landscapes," *Evolutionary Computation*, vol. 4, pp. 335-360, 1996.

[29] Z. Hua, B. Zhang, and L. Liang, "An approximate dynamic programming approach to convex quadratic knapsack problems," *Computers & Operations Research*, vol. 33, pp. 660-673, 2006.

[30] O. H. Ibarra and C. E. Kim, "Fast approximation algorithms for the knapsack and sum of subset problems," *Journal of the ACM*, vol. 22, pp. 463-468, 1975.

[31] A. M. Isen, T. E. Shalker, M. Clark, and L. Karp, "Affect, accessibility of material in memory, and behavior: a cognitive loop?" *Journal of Personality and Social Psychology*, vol. 36, pp. 1-12, 1978.

[32] B. A. Julstrom, "Greedy, genetic, and greedy genetic algorithms for the quadratic knapsack problem," *Genetic and Evolutionary Computation Conference*, Washington DC, USA, pp. 607-614, 2005.

[33] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," *IEEE International Conference on Neural Networks*, Perth, Australia, pp. 1942-1948, 1995.

[34] J. Kennedy, R. C. Eberhart, and Y. Shi, *Swarm Intelligence*. San Mateo, CA: Morgan Kaufmann, 2001.

[35] J. E. Laird, A. Newell, and P. S. Rosenbloom, "SOAR - an architecture for general intelligence," *Artificial Intelligence*, vol. 33, pp. 1-64, 1987.

[36] X. Li, J. Branke, and T. Blackwell, "Particle swarm with speciation and adaptation in a dynamic environment," *Genetic and Evolutionary Computation Conference*, Seattle, WA, USA, pp. 51-58, 2006.

[37] J. Liu, X.-L. Jin, and K. C. Tsui, *Autonomy Oriented Computing (AOC): From Problem Solving to Complex Systems Modeling*: Kluwer Academic Publishers, 2005.

[38] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. New York, USA: John Wiley & Sons, 1990.

[39] S. Martello, D. Pisinger, and P. Toth, "New trends in exact algorithms for the 0-1 knapsack problem," *European Journal of Operational Research*, vol. 123, pp. 325-332, 2000.

[40] R. Mendes, J. Kennedy, and J. Neves, "The fully informed particle swarm: Simpler, maybe better," *IEEE Transactions on Evolutionary Computation*, vol. 8, pp. 204-210, 2004.

[41] P. Michelon and L. Veilleux, "Lagrangean methods for the 0-1 quadratic knapsack problem," *European Journal of Operational Research*, vol. 92, pp. 326-341, 1996.

[42] J. K. Olick and J. Robbins, "Social memory studies: From "collective memory" to the historical sociology of mnemonic practices," *Annual Review of Sociology*, vol. 24, pp. 105-140, 1998.

[43] D. Pisinger, "Core problems in Knapsack algorithms," *Operations Research*, vol. 47, pp. 570-575, 1999.

[44] D. Pisinger, "Where are the hard knapsack problems?" *Computers and Operations Research*, vol. 32, pp. 2271-2282, 2005.

[45] D. Pisinger, A. Bo Rasmussen, and R. Sandvik, "Solution of large-sized quadratic knapsack problems through aggressive reduction," *INFORMS Journal on Computing*, vol. 19, 2007.

[46] D. J. Rader and G. J. Woeginger, "The quadratic 0-1 knapsack problem with series-parallel support," *Operations Research Letters*, vol. 30, pp. 159-166, 2002.

[47] A. Rodríguez and J. A. Reggia, "Extending self-organizing particle systems to problem solving," *Artificial Life*, vol. 10, pp. 379-395, 2004.

[48] R. Storn and K. V. Price, "Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optimization*, vol. 11, pp. 341-359, 1997.

[49] G. Syswerda, "Uniform crossover in genetic algorithms," *International Conference on Genetic Algorithms*, San Mateo, CA, pp. 2-9, 1989.

[50] N. A. Taatgen, *Learning without limits: from problem solving towards a unified theory of learning*, Ph.D. thesis, University of Groningen, the Netherlands, 1999.

[51] G. Tao and Z. Michalewicz, "Inver-over operator for the TSP," *International Conference on Parallel Problem Solving from Nature*, Amsterdam, the Netherlands, pp. 803-812, 1998.

[52] D. Weyns and T. Holvoet, "On the role of environments in multiagent systems," *Informatica*, vol. 29, pp. 409-421, 2005.

[53] D. Whitley, "An overview of evolutionary algorithms: practical issues and common pitfalls," *Information and Software Technology*, vol. 43, pp. 817-831, 2001.

[54] X.-F. Xie and W.-J. Zhang, "Solving engineering design problems by social cognitive optimization," *Genetic and Evolutionary Computation Conference*, Washington, USA, pp. 261-262, 2004.

[55] X.-F. Xie and W.-J. Zhang, "SWAF: Swarm algorithm framework for numerical optimization," *Genetic and Evolutionary Computation Conference*, Seattle, Washington, USA, pp. 238-250, 2004.

[56] X.-F. Xie and J. Liu, "A compact multiagent system based on autonomy oriented computing," *IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, Compiègne, France, pp. 38-44, 2005.

[57] X. Yao, "Global optimisation by evolutionary algorithms," *International Symposium on Parallel Algorithm/Architecture Synthesis*, Aizu-Wakamatsu, Japan, pp. 282-291, 1997.