STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LII**, Number 2, 2007

COMPONENT-BASED ANT SYSTEM FOR A BIOBJECTIVE ASSIGNMENT PROBLEM

CAMELIA-M. PINTEA AND ANDREEA VESCAN

Abstract. The paper proposes a component-based approach for a particular biobjective assignment problem: the Airport Gate Assignment Problem (AGAP). ACS-QAP [2] is the starting point for the proposed ACS model for solving an over-constrained version of AGAP, seeking feasible flight-to-gate assignments so that total passenger connection time, as proxied by walking distances, is minimized.

1. Introduction

Since the late 90's Component Based Development (CBD) is a very active area of research and development. CBSE covers both component development and system development with components [6]. There is a slight difference in the requirements and business ideas in the two cases and different approaches are necessary. Of course, when developing components, other components can be (and often must be) incorporated and the main emphasis is on reusability. Development using components is focused on the identification of reusable entities and relations between them, starting from the system requirements.

The complex biobjective problem modeled for an ant system algorithm using components is the over-constrained *Airport Gate Assignment Problem (AGAP)*. The problem has two objectives. The first one to minimize the number of flights assigned to the apron, when the number of flights exceeds the number of gates. The second objective is to minimize the total distance walk of the passengers in the airport.

The preliminary sections of the paper show the specifications of $AGAP$ and the techniques used to solve the specified problem. The main sections of the paper show the component-based solution of the ant system algorithm for solving $AGAP$ including the control flow and data flow.

Received by the editors: 30.08.2007.

2000 *Mathematics Subject Classification.* 68N30, 68T20.

1998 *CR Categories and Descriptors.* code I.6.5 [**Simulation and modeling**]: Subtopic – *Model Development*; code I.2.8 [**Artificial intelligence**]: Subtopic – *Problem Solving, Control Methods, and Search* .

2. The Biobjective Assignment Problem

A particular case of a *Quadratic Assignment Problem (QAP)* it is considered: *the Airport Gate Assignment Problem (AGAP).*

There were several attempts to solve *AGAP*. We are mentioning a Tabu Search metaheuristic by Xu and Bailey [17]. Another algorithm for solving *AGAP* was proposed by Ding et al. [7] with a greedy algorithm minimizing ungated flights while providing initial feasible solutions followed by a new neighborhood search technique.

The gate assignment problem has the objective of minimizing distance costs of the over constrained gate assignment problem, minimizing the number of ungated aircrafts and the total walking distances.

We consider the notations as in [7]:
N: set of flights arriving at the airport and/or departing from the airport;
M: set of gates available at the airport;
n: total number of flights, i.e., $|N|$, where $|N|$ denotes the cardinality of N;
m: total number of gates, i.e., $|M|$;
a_i: arrival time of flight i;
d_i: departure time of flight i;
w_{kl}: walking distance for passengers between the gates k and l;
f_{ij}: the number of passengers transferring between two flights i and j;

Two dummy gates are used: gate 0 the entrance or exit of the airport and gate $m+1$ the apron where flights arrive at when no gates are available. $y_{i,k}$ denotes that flight i is assigned to gate k if $y_{i,k}=1$ and otherwise $y_{i,k}=0$, where $(0<k<m+1)$.

$w_{k,0}$ is the walking distance between gate k and the airport entrance or exit. $f_{0,i}$ is the number of originating departure passengers of flight i. $f_{i,0}$ is the number of the disembarking arrival passengers of flight i. The walking distance between the apron and gate k is $w_{m+1,k}$.

The mathematical model of the biobjective problem: *The Airport Gate Assignment Problem* is following.

1. Minimize the number of flights assigned to the apron:

$$\sum_{i=1}^{n} y_{i,m+1} \rightarrow min,$$

2. Minimize the total walking distance:

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{m+1}\sum_{l=1}^{m+1} f_{i,j}w_{k,l}y_{i,k}y_{j,l}+$$

$$+\sum_{i=1}^{n}\sum_{l=1}^{m+1} f_{0,i}w_{0,l} + \sum_{i=1}^{n}\sum_{l=1}^{m+1} f_{i,0}w_{l,0} \to min,$$

The constraints of $AGAP$ are following.

$$\sum_{k=1}^{m+1} y_{i,k} = 1(\forall i, 1 \le i \le n) \tag{1}$$

These constraints ensure that every flight must be assigned to one and only one gate or assigned to the apron.

$$a_i < d_i(\forall i, 1 \le i \le n) \tag{2}$$

Constraint (2) specifies that each flight's departure time is later than its arrival time.

$$y_{i,k}y_{j,k}(d_j - a_i)(d_i - a_j) \le 0(\forall i, j, 1 \le i, j \le n, k \ne m+1) \tag{3}$$

Constraint (3) says that two flights schedule cannot overlap if they are assigned to the same gate.

$$y_{i,k} \in \{0,1\}(\forall i, 1 \le i \le n, \forall k, 1 \le k \le m+1) \tag{4}$$

The condition (4) disallows any two flights to be scheduled to the same gate simultaneously except if they are scheduled to the apron.

2.1. **The over-constrained approach.** For the over-constrained $AGAP$, the first step is to minimize the number of flights that need be assigned to the apron. The minimal number of flights can be computed by a greedy algorithm described in [7].

First of all the flights are sorted by the departure time and after that flights are assigned one by one to the gates. A flight is assigned to an available gate with latest departure time. If there are no gates available, the flight will be assigned to the apron.

The solution of the greedy algorithm is the optimal number of flights that can be scheduled in gates and it is used to provide initial feasible solutions for the *ACS*-based algorithm.

3. Ant System for Airport Gate Assignment Problem

The algorithm proposed is an improved version of *Ant Colony System (ACS)* for *Quadratic Assignment Problem (QAP)* [2]. The new algorithm is called *Reinforcing Ant System-QAP (RAS-QAP)* where the trail intensity is locally updated using the inner rule [14] (*local_update_pheromone_trails()*).

The problem of the *Airport Gate Assignment* is about finding the feasible flight-to-gate assignments so that total passenger connection time is minimized. The function we have to minimize is using the distances from check-in

to gates in the case of embarking or originating passengers, from gates to check-out in the case of disembarking or destination passengers and from gate to gate in the case of transfer or connecting passengers.

When the number of aircraft exceeds the number of available gates, in the over-constrained case, the distance from the apron to the terminal for aircraft assigned to these areas is also considered.

First are computed the distance potentials and flow potentials as in [17, 7]. Each edge (i, j), at moment t, is labeled by a trail intensity $\tau_{ij}(t)$.

Algorithm 1 RAS-QAP

```
assign_initial_pheromone_levels();
compute_distance_potentials();
compute_flow_potentials();
place_ants_on_locations();
for (t = 1) to t_max do
  for (k = 0) to num_ants − 1 do
    build_solution_for_ant(k);
    local_update_pheromone_trails();
    compute_cost_solution_for_ant(k) based on constraints (1)-(4);
    if ants[k].cost_solution < Best_solution.cost_solution then
      Best_solution=ants[k].cost_solution;
      Best_solution_t=t;
    end if
  end for
  update_pheromone_trails();
  write_experimental_results(t);
end for
```

Initially the ants are randomly placed in the graph nodes. At each iteration an ant moves to a new node. When an ant decides which node is the next move it does so with a probability based on the distance to that node and the amount of trail intensity on the connecting edge. Evaporation takes place, at each step, to stop the intensity trails increasing unbounded.

Two tabu lists are used. The first tabu list stores the ants visited locations, so that they never visit them again. The second tabu list stores the activities that have been mapped to the visited locations.

To favour the selection of an edge that has a high pheromone value and high visibility value, a probability is considered. Compute cost solution for ant k and update the best solution. The global update rule is applied to the edges belonging to the *best tour-solution*.

The solution of the algorithm is the tour with the minimal cost. A solution is a vector with the potentially best distances, called distance potentials, and potentially best flows, called flow potentials. The algorithm runs for a given number of iteration t_{max}.

4. Component elements

A system can be designed and implemented by assembling components, customizing or extending them as needed; and publishing components in a form that can be applied to design and construct others, based purely on interface specifications.

One of the most popular definitions of a component was offered by a working group at ECOOP (European Conference on Object-Oriented Programming).

Definition 1. *A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and it is subject to composition by third parties.* [15]

Clemens Szyperski and David Messerschmitt [15] give the following five criteria for what a software component shall be to fulfill the definition: multiple-use; non-context-specific; composable with other components; encapsulated i.e., non - investigable through its interfaces and a unit of independent deployment and versioning.

We must first establish our entities involved in the component system definition before describing our component-based approach for modeling AGAP.

Considering X a component over the set A of attributes, we will use the following notations: $inports(X) \in A$ - represents the set of input ports (attributes) of the component X; $outports(X) \in A$ - represents the set of output ports (attributes) of the component X; $attributes(X) \in A$ - represents the set of attributes of the component X.

We can view components from a different perspective [10] as simple components and compound components with the following characteristics:

- **Simple Component** - over A is a 5-tuple **SC** of the form

  $(inports, outports, attributes, function, \prec_{SC}), where$:

  - $inports$ is a n-tuple $(in_1, ..., in_n)$ of attributes;
  - $outports$ is a m-tuple $(out_1, ..., out_m)$ of attributes and $(out_1, ..., out_m) \notin inports(SC)$;
  - $function$ is an n-ary function $Type(in_1) \times Type(in_2) \times ... \times Type(in_n) \rightarrow Type(out_1) \times Type(out_2) \times ... \times Type(out_m)$.

– *attributes* is defined to be the set of attributes consisting of the inports and the outports;
– the binary relation
$\prec_{SC} \subseteq (inports(SC) \times outports(SC)) \times outports(SC)$.

- **Compound component** - over A is a group of connected components, in which the output of a component is used as input by another component from this group.

A graphical representation of our view of components is given in Figure 1.

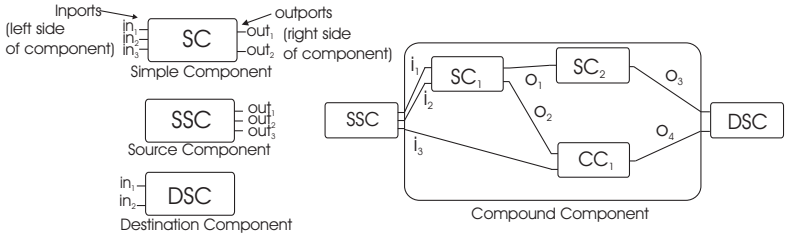

FIGURE 1. Components graphical representation. The compound component contains two simple components SC_1 and SC_2 and one compound component CC_1

Two particular components are the source component [1] and the destination component [2]. The source component represents the "read" component and the destination component represents the "write" component, components that should exists in any software system. In our approach every assembly system (and subsystem) has only one source component and only one destination component.

4.1. **Component construction and execution elements.** The wiring of components in order to construct a component-based system is made using a connection between the output of a component and the input of another component.

A **connection K** is made of an origin - output of a component, and a destination - input of another component:

$$K = (origin, destination),$$

[1]source component has no inports and generates data provided as outports in order to be processed by other components

[2]destination component has no outports and receives data from its inports and usually displays it, but it does not produce any output

where *origin* is an *outport* of a component; *destination* is an *inport* of a component.

The composition result is also a component, a compose component using [10] notation. The resulted system is called **BlackBox** and is specified as follows:

$$BlackBox = (\{in\}, \{out\}, \{component\}, \{connection\}),$$

where *in* represent the inputs for the blackbox; *out* represent the output of the blackbox; *component* represent the components involved in the composition and *connection* represent all the connections between the involved components.

The execution of the BlackBox component is composed of sequences of the form:

$$(Op_0, C_0), (Op_1, C_1), (Op_2, C_2), ...$$

where for each $i \geq 0$, Op_i is a subset of possible operations and C_i is a subset of components ready for execution.

The possible operations are:

- propagation - this rule moves values that have been generated by a component along connections from the component's outport to other components;
- evaluation - the component function is evaluated and the result is passed to the output of the component.

State of execution. At a given time of execution, the state is presented as follows:

$$State = (\{operation\}, \{componentForEval\}),$$

where $operation = \{C->, C=\}$;

- $C->$ - propagation operation from component C;
- $C=$ - evaluation operation of component C.

componentForEval - a component ready for evaluation.

If at a given time, both types of operation can be performed, the propagation operation is chosen. Between many evaluation operations, one component is chosen randomly.

4.2. **Component assembly construction process.** A top-down approach is used when reasoning about the way to solve a problem (from the system requirements develop the needed modules to accomplish the requirements of the system under development) and a bottom-up approach when assembling the pieces in order to build the desired final system [16].

The components composition is accomplished using a bottom-up approach: starting from a given set of components (stored in a repository) there are two main steps to obtain the final system:

(1) newly obtained components (if necessary) by assembling given components (simple components and/or compound components);
(2) compose the final system from the new set of available components.

Reasoning in a top-down approach we refer to the second step (from the bottom-up approach) as the first hierarchical level of the final system(s) and to the first step (which may contain many inside steps to develop new components) as intermediary hierarchical levels of the system.

5. Component-based Ant System for Airport Gate Assignment Problem

This section presents the architecture of the component-based approach for the AGAP, the control flow and the data flow model. At the end of this section we show how the computation steps are successively executed.

5.1. **AGAP architecture.** In section 3 we have presented the ant system solution for the Airport Gate Assignment Problem. Based on the pseudocode algorithm we can describe the solution using components as in Figure 2 as a first level of design: initialization, computation and printing the obtained results. The next two levels (decompositions) are also presented.

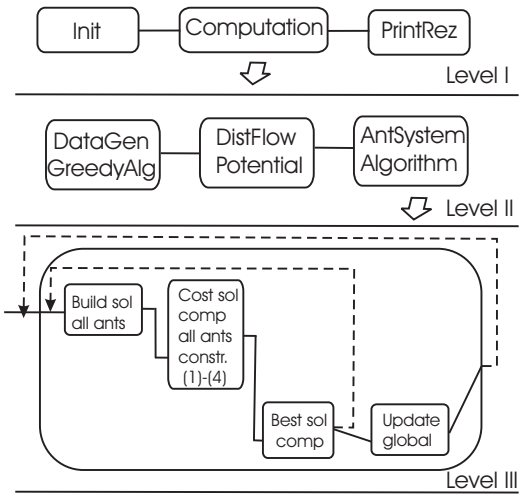

Figure 2. Architectural levels

The second level contains the data generation for the greedy algorithm, the computation of the distance flow potentials and the ant system algorithm.

The third level contains a more detail view of the ant system algorithm computation: build solution (for all ants), then cost solution computation (for

all ants) based on the constraints, best solution computation (for all ants) and the update global rule computation.

All these computations are performed for t_{max} steps/times. The build solution computation, cost solution and best solution computation are all computed for all ants, thus the loop is executed until the number ants *num_ants* (see algorithm 1 for details) is reached.

5.2. **Control flow AGAP model.** Figure 3 shows the overview of the system and the control flow.

The

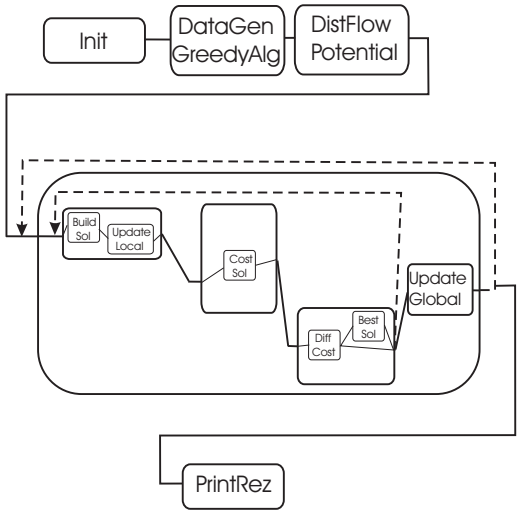

FIGURE 3. Control flow component-based RAS-QAP system

5.3. **Data flow AGAP model.** Figure 4 shows the overview of the system with all the components from all the architecture levels - the data flow. Component *DataGenGreedyAlg* corresponds to the first statement in algorithm 1, component *DistFlowPotential* to the second, third and fourth statements and component *AntSystemAlgorithm* to the rest of the statements.

FIGURE 4. Data flow component-based RAS-QAP system

The data transmitted from the init to the greedy algorithm is that from the problem description: N - set of flights arriving at (and/or departing from) the airport, M - set of gates available at the airport, n - total number of flights and m - total number of gates.

The greedy algorithm gives the optimal number of flights that can be scheduled in gates.

The $DistFlowPotential$ computes the sums based on the number of passengers f_{ij} and the walking distance from gate k to gate l, w_{kl}.

The $AntSystem$ computes L^+, thus minimizing the total walking distance.

5.4. **General and internal computation steps.** The steps of the computation of the ant colony component model for $AGAP$ are described successively.

We denote each component from the architecture (see figure 2 and figure 3 for details) with the following acronyms in order to be more easier to read: $Init$ component with I; $DataGenGreedyAlg$ with $DGGA$; $DistFlowPotential$ with DFP; $BuildSol$ and $UpdateLocal$ with $BSUL$; $CostSol$ with CS; $DiffCost$ and $BestSol$ with $DCBS$; $UpdateGlobal$ with UG and $PrintRez$ with PR.

General computation steps for the representation from figure 4:

- $state_0 = (\{I \equiv\}, \{I\})$;

-N, M, n, m receives the initial values.

- $state_1 = (\{I \rightarrow\}, \{\})$;

-data is propagated through the connections to the $DGGA$ component: N, M, n, m.

- $state_2 = (\{DGGA \equiv\}, \{DGGA\})$;

-the data are generated in the corresponding intervals for a_i, d_i, w_{kl} and f_{ij}. -the greedy algorithm minimizes the number of flights assigned to the apron and returns the $flights$ array.

- $state_3 = (\{DGGA \rightarrow\}, \{\})$;

-$DGGA \rightarrow$: data is propagated through the connections to the DFP component.

- $state_4 = (\{DFP \equiv\}, \{DFP\})$;

-$DFP \equiv$: computes the sums based on the number of passengers f_{ij} and the walking distance w_{kl} from the gate k to gate l.

- $state_5 = (\{DFP \rightarrow\}, \{\})$;

-$DFP \rightarrow$ data is propagated through the connection to the input of the ASA component.

- $state_6 = (\{ASA \equiv\}, \{ASA\})$;

-$ASA \equiv$ minimizes the total walking distance.

- $state_7 = (\{ASA \rightarrow\}, \{\})$;

-$ASA \rightarrow$ data is propagated through the connection to input of the PR component.

- $state_8 = (\{PR \equiv\}, \{PR\})$;

-$PR \equiv$ prints the obtained result.

- $state_9 = (\{\}, \{\})$.

-There are no more possibilities of applying either propagation or evaluation.

-The execution of the components involved in the system is finished.

The execution states for the representation from figure 3 (only Level III) are described in the following. The execution of the components starts from $state_6$ from the general computation from above.

- $state_{6_1} = (\{BSUL \equiv\}, \{BSUL\})$; $state_{6_{1'}} = (\{BSUL \rightarrow\}, \{\})$;
- $state_{6_2} = (\{CS \equiv\}, \{CS\})$; $state_{6_{2'}} = (\{CS \rightarrow\}, \{\})$;
- $state_{6_3} = (\{DCBS \equiv\}, \{DCBS\})$; $state_{6_{3'}} = (\{DCBS \rightarrow\}, \{\})$;

-if the previous steps were not executed for each ant then the state 6_1 is following, else state 6_4.

- $state_{6_4} = (\{UG \equiv\}, \{UG\})$; $state_{6_{4'}} = (\{UG \rightarrow\}, \{\})$.

-if the maximum t_{max} number of iterations are not reached then the state 6 is following, else state 7.

6. Conclusions

Ant algorithms are based on the real world phenomena that ants are able to find their way to a food source and back to their nest, using the shortest route. A component based Ant System for the Airport Gate Assignment Problem is introduced. The way of using the components is shown: the control flow and the data flow. The model execution steps are also illustrated.

References

[1] O. Babic, D. Teodorovic, V. Tosic, *Aircraft stand assignment to minimize walking*, Journal of Transportation Engineering, 110, pp. 55-66, 1984.

[2] A. Barton, *A simplified Ant Colony System applied to the Quadratic Assignment Problem*, Technical Report National Research Council of Canada no.47446, 2005.

[3] J. Braaksma, J. Shortreed, *Improving airport gate usage with critical path method*, Transportation Engineering Journal of ASCE 97, pp. 187-203, 1971.

[4] Y. Cheng, *Network-based simulation of aircraft at gates in airport terminals*, Journal of Transportation Engineering, pp. 188-196, 1998.

[5] Y. Cheng, *A rule-based reactive model for the simulation of aircraft on airport gates*, Knowledge-based Systems, 10, pp. 225-236, 1998.

[6] I. Crnkovic, *Component-based Software Engineering - New Challenges in Software Development*, Software Focus, 2001.

[7] H. Ding, A. Lim, B. Rodrigues, Y. Zhu, *The airport gate assignment problem*, hicss, p.30074b, Proceedings of the 37th Hawaii International Conference on System Sciences (HICSS'04), pp. 74-81, Track 3, 2004.

[8] M. Dorigo and L. M. Gambardella, *Ant Colony System: A cooperative learning approach to the Traveling Salesman Problem*, IEEE Trans. Evol. Comp., 1:5366, 1997.

[9] M. Dorigo, *Optimization, Learning and Natural Algorithms (in Italian)*. Ph.D thesis, Dipartamento di Elettronica, Politecnico di Milano, Italy, pp.140, 1992.
[10] A. Fanea, S. Motogna, *A Formal Model for Component Composition*, Proceedings of the Symposium Zilele Academice Clujene, pp. 160-167, 2004.
[11] A. Haghani, M. Ching Chen, *Optimizing gate assignments at airport terminals*, Transportation Research, 32(6), pp. 437-454, 1998.
[12] T. Obata, *The quadratic assignment problem: Evaluation of exact and heuristic algorithms*, Tech. Report TRS-7901, Rensselaer Polytechnic Institute, Troy, New York, 1979.
[13] V. Maniezzo, A. Colorni, M. Dorigo, *The Ant System applied to the Quadratic Assignment Problem*, Technical report 94/28, IRIDIA, Université de Bruxelles, Belgium, 1994.
[14] C-M. Pintea, D.Dumitrescu, *Improving ant systems using a local updating rule*, Proc. 7-th Int. Symposium on Symbolic and Numeric Algorithms for Scientific Computing, IEEE C.S.Press, pp. 295-299, 2005.
[15] C. Szypersky, *Component Software: Beyond Object-Oriented Programming*, Addison-Wesley, 1998.
[16] A. Vescan, S. Motogna, *Syntactic automata-based component composition*, The 32nd EUROMICRO Software Engineering and Advanced Applications (SEAA), Proceeding of the Work in Progress session, ISBN 3-902457-11-2, 2006.
[17] J. Xu, G. Baile, The Airport Gate Assignment Problem: Mathematical Model and a Tabu Search Algorithm, *HICSS*, 34th Annual Hawaii International Conference on System Sciences, Vol.3, 2001.
[18] S. Yan, C.-M. Chang, *A network model for gate assignment*, Journal of Advanced Transportation, 32(2), pp. 176-189, 1998.

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania
E-mail address: {cmpintea, avescan}@cs.ubbcluj.ro