

Optimal work assignment in team processes for maximizing cooperation

Akhil Kumar
Smeal College of Business
Penn State University
University Park, PA 16802, USA
AkhilKumar@psu.edu

Abstract

Resource management in teams and workflows deals with assignment of tasks to workers or agents. In team formation, it is necessary to ensure that members of a team are compatible with each other. When an instance such as a workflow of an insurance claim process flows through an organization, various agents perform different tasks in this process instance. Here it is necessary to make sure that the agents who participate in this instance are compatible in order to speed up the processing of the instance. Of course, the degree of required cooperation between various participants varies and this should also be considered. The contribution of this paper is to develop a model to capture the compatibility between agents while assigning work in a workflow system or a team. The model is also tested through a simulation and the results are compared with the results from a heuristic algorithm. The optimal solution is typically about 20% better than the heuristic one, and in some cases it is more than 40% better. An algorithm for speeding up the solution procedure through partitioning, and extensions to the model are discussed.

Keywords: resource management, team assignment, cooperation, optimization.

1 Introduction

Much work within organizations takes place in teams whether it is designing a car, performing surgery or processing a customer's insurance claim application. Naturally, it is very important that members of a team, in addition to having the requisite qualifications, also be compatible with one another in order to ensure smooth execution and flow of the work. Of course, in a team of n workers or agents, it is not necessary that every pair of members must be fully compatible with each other, but the goal in general would be to maximize overall compatibility particularly across agents whose roles requires considerable collaboration and cooperation. Non-cooperation can result in loss of productivity. In a similar vein, the need for

optimization also arises in business processes. In a typical insurance claim process, several tasks must be done by different people in a certain order. After a worker or agent completes her task she hands off the process workflow to the next agent. In a *hard handoff* no further interaction between the two agents may be required. But more often the handoffs are *soft*, i.e. the two agents may need to interact later for queries and clarifications. Hence, cooperation is necessary between the two agents so that the work can be performed smoothly. In general, an agent doing a later task in a workflow may need to refer back to consult with an agent who did a previous task for the same case. Hence, there is considerable need for cooperation between agents who perform different tasks on the same case or process instance.

Workflow management systems can be viewed from various perspectives such as: control flow, data flow and resource modeling. The control flow [1, 2] describes the ordering relationships between various tasks. The data flow [16] describes the data input needs of each task and the output data produced by it. The resource model [9,20] refers to the roles and specific agents who are qualified to perform various tasks. Another aspect of the resource model is that since agents are also a resource, tasks must be assigned to them in a suitable way. Most resource assignment algorithms consider issues like suitability, urgency, conformance and availability [8, 11] while allocating tasks to agents. However, they fail to recognize the interactions between the agents performing tasks in a workflow. Thus, for a given task instance that involves various steps, say for insurance claim processing or travel expense processing, etc. there is need for interaction between agents who perform various steps.

Hence, the execution of a process instance is really a team effort involving multiple handoffs and there is a need for making the handoff as smooth as possible. If Joe performs a task A for an insurance claim and hands it to Sue who does task B, there is a chance that Sue may

need to refer back to Joe for various clarifications. If Joe and Sue get along well with each other as co-workers, they will respond to each other's queries by email or phone more willingly, while if they do not get along they will tend to procrastinate. Hence, there is a need for a notion of compatibility between agents, to denote how well two workers tend to cooperate with one another, and this should be considered while assigning tasks to agents. Of course, cooperation is not always required among all members of a team, and degrees of cooperation vary. So we need a model that can allow us to specify required degree of cooperation between any pair of individuals through a cooperation matrix.

In addition to the need for cooperation in a variety of business applications, the medical domain is another area where multiple roles must work together in order to achieve an outcome, and compatibility and smooth handoffs between various personnel involved (such as doctors, nurse, lab technicians) in patient care is very important [5,10]. In this paper we show how to model compatibility between agents while making work assignments so as to achieve a high degree of overall compatibility for the process. Section 2 gives a basic framework and preliminaries. Then, Section 3 describes our model for maximizing compatibility. Next, Section 4 gives experimental results comparing the optimal solution with those from a greedy heuristic, while Section 5 develops an algorithm for solving the optimization model faster by decomposing the problem formulation into subproblems. Section 6 gives variants of our model to extend it to different situations. Later, Section 7 shows how to calculate compatibilities between agents using logs and also discusses related work. Finally section 8 concludes the paper.

2 Basic Framework

Consider an example of an insurance claim process (see Figure 1). In this process, a claim is received, and then *checked* by a reviewer who verifies that it is a valid claim. Next, it is

evaluated by an adjuster who determines the amount of the settlement. A manager must *approve* the claim, and finally the accounts officer issues a *payment* for it. Thus, the key steps or tasks, the roles who perform each step and the agents in the roles in this process are:

Receive claim (role: customer service rep; agents: John, Mary)

Review and validate claim, and assign to adjuster (role: reviewer; agents: Beth, Sue)

Evaluate claim and determine settlement (role: adjuster; agents:Mike, Jim)

Approve payment (role: manager; agents: Jen, Pat)

Make payment (role: accounts officer; agents: Mark, Lin)

Moreover, one should also consider the need for interaction between various roles that perform the tasks. From one perspective it may be argued that each task is independent and once an agent in a role has performed her task, she does not need to worry about this process instance. On the other hand, there may be also much need for interaction as depicted by the dotted arrows in Figure 1 where the tasks and the corresponding roles for each task are shown. After a claim is received, the reviewer could refer back to the customer service rep for clarification about missing information on the claim. Alternatively, it may occur that the information about the location of an accident is incomplete and the rep may need to call the customer and get more details. Similarly, the adjuster may need to consult with the reviewer for additional details. Finally, the manager could seek clarifications with the adjuster regarding the amount recommended before approving the payment.

Thus, even though formal representations for workflow processes may not actually show the details, there is very often a need for such referrals. In this sense the handoffs between tasks in real world processes are fuzzy as opposed to crisp and there is considerable need for cooperation between various agents who perform different tasks for a process instance such as this one. But formal depictions tend to gloss over them or abstract from them. Our goal is to

capture notions of compatibility between agents who will perform tasks where handoffs are important. Hence, a metric for compatibility is required.

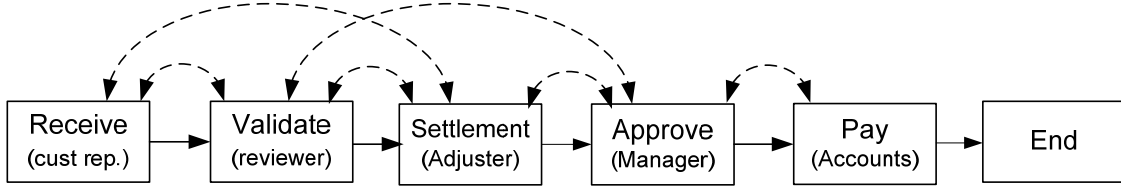


Figure 1: A simplified insurance claim process with several tasks and roles (dotted lines show cooperation among tasks)

Our metrics for compatibility within a team or a process workflow are:

$$Total\ Compatibility = \sum_{u1,u2,t1,t2} fit_{u1,u2,t1,t2} * coop_{t1,t2} * cweight_{u1,u2}$$

$$Average\ Compatibility = \frac{Total\ Compatibility}{\sum_{t1,t2} coop_{t1,t2}}$$

Where

$$fit_{u1,u2,t1,t2} : \begin{cases} 1 & \text{if user } u1, u2 \text{ perform tasks } t1, t2 \text{ respectively} \\ 0 & \text{otherwise} \end{cases}$$

$$coop_{t1,t2} : \begin{cases} 1 & \text{if cooperation is required between the agents of tasks } t1 \text{ and } t2 \\ 0 & \text{otherwise} \end{cases}$$

$cweight_{u1,u2}$: compatibility between users $u1, u2$ on a continuous scale of 0 – 1

The fit and cooperation values are stored in two matrices. Table 1 is an agent-agent compatibility matrix with values on a scale of 0 to 1 (from low to high compatibility). Table 2 gives a binary cooperation matrix for all pairs of tasks, where a '0' means cooperation between a pair of tasks is not required, and '1' that it is. The end task is not shown in any table because it is an automated task and is not performed by a human.

Example 1: (partial cooperation) Below we calculate average compatibilities using the values in Tables 1 and 2. There are five main tasks (ignoring the End task) in this instance.

Moreover there are 6 interactions where cooperation is required as per Table 2. Clearly, several combinations of agent assignments are possible here. Let us look at two examples:

Assignment 1 (heuristic):

(cust rep: John ; Reviewer: Sue; Adjuster: Jim; Manager: Pat; accounts officer: Mark)

Average compatibility = $(0.1 + 0.3 + 0.6 + 0.6 + 0.1 + 0.1)/6 = 0.3$

Assignment 2 (optimal):

(cust rep: Mary; Reviewer: Beth; Adjuster: Jim; Manager: Jen; accounts officer: Mark)

Average compatibility = $(0.3 + 0.8 + 0.8 + 0.7 + 0.8 + 0.7)/6 = 0.683$

Clearly, there is a large difference (of more than 100%) in average compatibility between these two selections.

Table 1: agent-agent Compatibility matrix (*cweight*)

Role > (Task) >	Cust. Rep (receive)		Reviewer (validate)		Adjuster (settle)		Manager (approve)		Accounts (pay)	
	John	Mary	Beth	Sue	Mike	Jim	Jen	Pat	Mark	Lin
John	-	-	0.8	0.1	0.8	0.3	0.9	0.3	0.4	0.2
Mary	-	-	0.3	0.7	0.2	0.8	0.9	0.2	0.1	0.8
Beth	-	-	-	-	0.3	0.8	0.7	0.3	0.2	0.9
Sue	-	-	-	-	0.9	0.6	0.4	0.6	0.8	0.4
Mike	-	-	-	-	-	-	0.3	0.9	0.8	0.1
Jim	-	-	-	-	-	-	0.8	0.1	0.3	0.9
Jen	-	-	-	-	-	-	-	-	0.7	0.3
Pat	-	-	-	-	-	-	-	-	0.1	0.8

Table 2: Cooperation matrix (partial cooperation required)

	Validate	Settle	Approve	Pay
Receive	1	1	0	0
Validate	-	1	1	0
Settle		-	1	0
Approve			-	1

Example 2: (full cooperation) Next consider a variation of the above example. Instead of assuming that cooperation between some pairs of participants is necessary, let us assume that all participants who work on an instance of a process must cooperate with each other. The

corresponding cooperation matrix is shown in Table 3. In this case the heuristic and optimal assignments are as follows:

Assignment 3 (heuristic):

cust rep: Mary ; Reviewer: Sue; Adjuster: Jim; Manager: Jen; accounts officer: Lin

Average compatibility = $(0.7 + 0.8 + 0.9 + 0.8 + 0.6 + 0.4 + 0.4 + 0.8 + 0.9 + 0.3)/10 = 0.66$

Assignment 4 (optimal):

cust rep: Mary; Reviewer: Beth; Adjuster: Jim; Manager: Jen; accounts officer: Lin

Average compatibility = $(0.3 + 0.8 + 0.9 + 0.8 + 0.8 + 0.7 + 0.9 + 0.8 + 0.9 + 0.3)/10 = 0.72$

Table 3: Cooperation matrix (full cooperation required)

	Receive	Validate	Settle	Approve	Pay
Receive	-	1	1	1	1
Validate		-	1	1	1
Settle			-	1	1
Approve				-	1
Pay					-

In this example, it turns out that the difference between the optimal and the heuristic solution is much smaller than in the previous one.

We have considered two scenarios involving different levels of cooperation. In general, the cooperation matrix could vary, and the best assignment will also be different accordingly. Next we describe our model for finding an optimal solution so as to maximize cooperation within the team.

3 Model

The objective of the model is to maximize total (or average) *compatibility*. However, we express our objective function so as to minimize total *incompatibility* and the reason for this is explained shortly. Our notion of overall compatibility is the sum of all pair-wise compatibilities between agents who are involved in task-pairs of a process that require cooperation. Since pair-

wise agent-agent compatibility ranges between 0 and 1, $incompatibility = (1 - compatibility)$.

Our model is given below:

$$Minimize \sum_{u1,u2,t1,t2} fit_{u1,u2,t1,t2} * (1 - cweight_{u1,u2})$$

Subject to:

$$\sum_u does_{u,t} = 1, \forall t \quad (1)$$

$$does_{u,t} \leq cando_{u,t} \quad (2)$$

$$does_{u1,t1} + does_{u2,t2} - fit_{u1,u2,t1,t2} \leq 1, \forall t1, t2 \text{ where } coop(t1, t2) = 1 \quad (3)$$

$$avail_u > cutoff1 \quad (4)$$

Where:

$$fit_{u1,u2,t1,t2} = \begin{cases} 1 & \text{if user } u1, u2 \text{ perform tasks } t1, t2 \text{ respectively} \\ 0 & \text{otherwise} \end{cases}$$

$$does_{u,t} = \begin{cases} 1 & \text{if user } u \text{ does task } t \\ 0 & \text{otherwise} \end{cases}$$

$$cando_{u,t} = \begin{cases} 1 & \text{if user } u \text{ is qualified for task } t \\ 0 & \text{otherwise} \end{cases}$$

$$coop_{t1,t2} = \begin{cases} 1 & \text{if cooperation is needed between } t1 \text{ and } t2 \\ 0 & \text{otherwise} \end{cases}$$

$cweight_{u1,u2}$: compatibilty between users $u1, u2$

$avail_u$: availability of user u

Constraint 1 says that every task must be assigned to exactly one agent. The second constraint requires that the agent should be qualified to do this task (i.e. she *can do* the task). The third constraint forces the fit variable between two agents doing tasks that have a handoff between them to 1. Thus, if there is a handoff requirement between tasks, $t1$ and $t2$, and agents $u1$ and $u2$ perform them respectively, then the corresponding fit variable $f_{u1,u2,t1,t2}$ must be 1. Note that here we are not interested in the compatibility between the agents who perform independent tasks that do not require any cooperation between them.

Now, the reason the objective function minimizes total incompatibility is as follows: If we try to maximize compatibility then the *fit variables are all forced to 1* incorrectly resulting in

an incorrect formulation and thus it produces a wrong solution. However, if we express the objective as minimizing incompatibility this error does not occur, and a fit variable $fit_{u1,u2,t1,t2}$ assumes a 1 value only when user $u1$ does task $t1$ and user $u2$ does task $t2$.

To construct a model like this we only need to know the data in the `coop`, `cweight` and `cando` matrices. The size of this problem is $O(u^2 t^2)$, where u is # of users and t is # of tasks.

3.1 A Greedy heuristic

Here we describe a Greedy heuristic to solve the agent allocation problem. The main steps are shown in Figure 2. The `coop`, `cweight` and `cando` arrays are taken directly from the model described above. The `assign` array is initialized to -1 at the start (lines 6-8). Next, for each successive task $t1$ (line 9), we consider each agent $u1$ (line 11) who can do $t1$. Then, for each agent $u1$ (line 11) and for every other task $t2$ (line 13) such that cooperation between $t1$ and $t2$ is required (line 14), we find the maximum compatibility agent with agent $u1$ from the `cweight` array (line 15) and accumulate the compatibility in a variable `score1` (line 16). This is repeated for every task $t2$ that requires cooperation with $t1$ (lines 13-18), and in this way, a score is computed for each agent who can perform task $t1$ (line 16). Finally, the user with the maximum score is assigned the task $t1$ (line 21). In this way, every task is assigned to an agent. The algorithm returns the `assign[]` array as the solution.

This is a greedy algorithm in that at each successive step we assign agents to a task based on the basis of the best compatibility for this particular assignment without optimizing across all tasks.

Algorithm Greedy

```
1  Input:
2  coop[][] , cweight[][] , cando[][]
3  Output:
4  assign[] matrix
5  Procedure:
6  For each (task t = 1, ..., num_tasks)
7  assign[t] = -1
8  End for
9  For each (task t1 = 1, ..., num_tasks)
10 tscore = 0
11   For each (u1 ∈ cando[i])
12     Score2(u1) = 0
13     For each (task t2 = t1+1, ..., num_tasks)
14       If (coop(t1,t2) == 1 && assign[t1] == -1 && assign[t2] == -1)
15         tscore = Max{cweight(u1,u2), u2 ∈ cando[t2]}
16         score1(u1) = score1(u1) + tscore
17       End If
18     End for
19   End for
20   best_agent = u* s.t. score1[u*] = Max{score1[u], u ∈ cando[t1]}
21   assign[t1] = best_agent
22 End for
23 Return(assign[])
24 End Procedure
```

Figure 2: A greedy heuristic for agent assignment

4 Results

4.1 Experimental Setting

We conducted experiments to test this approach and to compare the greedy heuristic with the optimal solution. The greedy heuristic was implemented in Python, while the optimal solution was found by solving the model using CPLEX software [6]. We used a simulation first to create the data for the experiments, and the parameters of the simulation are given in Table 4. In particular there are 10 tasks in the process and 20 agents. Each task can be done by either 2 or 3 agents. First, we select the number of qualified agents for each task (2 or 3, with equal probability), and then pick the actual agents at random. The cooperation model is: (1) each task i must cooperate with the next task $i+1$; (2) In addition, we also require that with probability 0.5 there should be cooperation between task i and each of tasks $i+1$ and $i+2$. Finally an agent-agent compatibility matrix is generated where compatibility values of 0.1, 0.2, ..., 0.9 are randomly

assigned. If the same agent is performing two tasks, then the compatibility is 0.99. In these experiments we assumed availability of all agents was 1, i.e. they were all available.

Table 4 : Parameters used in the simulation experiment

Parameter	Description	value
# tasks	Number of tasks	10, 20
Total # agents	Number of agents	20, 40
Task- agent assignment	For each task, assign agents who can perform the task	Pick 2 or 3 agents at random
Cooperation requirement	Between tasks i and $i+1$ with prob. 1, and between i , and, $i+2$ and $i+3$, with prob. 0.5	
Compatibility weight	Weight between 0 and 1 to measure degree of fit between two agents where handoff is important	0.1,0.2, ... 0.9
Availability	Extent of availability of an agent (0.0,...1.0)	1.0

4.2 Results 1

The results are given in Table 5, where we summarize 10 cases. For each case the actual agent assignment produced by the heuristic and the optimal algorithms are shown. Thus, for case 1, the heuristic produces an assignment where task 1 is assigned to agent 19, task 2 to agent 10, and so on. For this case the heuristic assignment is very surprisingly close to the optimal solution. In fact 9 out of 10 agent assignments are the same, with the exception that task 9 is assigned to agent 10 in the heuristic instead of agent 7. We also report the average compatibility, i.e. the average of the compatibility values across the '1' entries in the cooperation matrix, along with the percentage gap between the optimal and the heuristic solutions. In case 1, the heuristic is worse than the optimal by just about 6%, but in other cases, the gap is larger, even as high as 40% in case 9. Overall, across all 10 cases the average gap is about 19%.

In terms of running time, CPLEX ran all the 10 cases in under a second on a high end server machine. The results clearly show that it is useful to have an optimal solution since in some cases the gap can be large.

Table 5: Results comparing average compatibility for the heuristic and optimal solutions (10 tasks, 20 agents)

Case	Avg. Compatibility		% gap	Agent assignment	
	Heuristic	Optimal		Heuristic	Optimal
1.	0.656	0.700	6.29	[19 ,10, 9, 14, 5,10,3,2 ,7,1]	[19 ,10, 9, 14, 5,10,3,2 ,10,1]
2.	0.650	0.759	14.36	[16,17,3,1,4,12,9,2,4,15]	[4,13,3,4,4,12,9,1,12,5]
3.	0.669	0.760	11.97	[1,7,4,8,10,11,2,18,5,0,6]	[16,5,8,6,2,2,18,1,0,0]
4.	0.653	0.785	16.82	[19,18,13,7,1,16,10,19, 10,8]	[19,18,18,7,1,7,10,19,10, 8]
5.	0.591	0.740	20.14	[17,3,1,3,10,16,1,0,12, 12]	[17,3,1,3,10,3,1,13,12,12]
6.	0.615	0.737	16.55	[19 ,10, 9, 14, 5,10,3,2 ,7,1]	[19,4, 15,4,2,4,12,2,14,16]
7.	0.461	0.597	22.78	[10,9,0,1,12,2,11,8,7, 16]	[9,9,10,3,12,2,15,0,2,16]
8.	0.550	0.761	27.73	[18,13,9,6,13,8,9,4,14, 13]	[18,18, 15,6,18,8,8,4,18,13]
9.	0.466	0.780	40.26	[14,12,7,10,10,16,7,5,17,12]	[14,0,19,14,10,0,5,5,2,6]
10.	0.615	0.730	15.75	[12,7,5,1,1,2,16,14, 14,13]	[3,17,9,1,1,2,11,8,8,9]
Avg.	0.593	0.735	19.32		

4.3 Results 2

Next we describe a second experiment with 20 tasks and 40 agents. The results are given in Table 6. Again the results from the heuristic are compared with the optimal solution from CPLEX. Now there is an average gap of 17% between the performance of the optimal and the heuristic, and it lies between 8% (case 3) and 23% (case 7). In case 3, 6 out of 10 agent assignments are the same, while in case 7, 5 out of 10 are the same. However, even changing a small number of assignments can sometimes make a big difference highlighting the drawbacks of a greedy heuristic. Nevertheless, as compared to the previous set of results, there is less variability in these results. Again the running times for the CPLEX software were very small, around 1-2 seconds.

Table 6: Results comparing average compatibility for the heuristic and optimal solutions (20 tasks, 40 agents)

Case	Avg. Compatibility		% gap	agent assignment (only first 10 task assignments shown)	
	Heuristic	Optimal		Heuristic	Optimal
1.	0.615	0.684	10.09	[38 30 27 3 21 7 5 29 21 20 ...]	[27 3 22 3 21 7 35 29 21 20...]
2.	0.568	0.717	20.78	[24 23 20 5 21 19 24 23 31 2 ..]	[11 23 23 5 10 18 24 23 31 31]
3.	0.557	0.607	8.24	[21 8 25 22 17 23 21 29 7 30 ...]	[21 8 38 22 11 19 21 15 7 30..]
4.	0.605	0.759	20.29	[23 34 12 24 19 25 5 29 13 22]	[23 34 12 0 19 1 5 29 13 22..]
5.	0.608	0.712	14.61	[34 17 13 24 35 14 35 10 31 1]	[34 0 2 35 35 19 35 10 31 1..]
6.	0.596	0.771	22.70	[15 38 4 35 12 28 24 31 34 24]	[0 23 4 4 10 10 18 31 2 24...]
7.	0.567	0.734	22.75	[27 31 18 29 9 27 12 7 12 10]	[27 31 4 29 6 31 12 31 28 10..]
8.	0.556	0.718	22.56	[18 7 13 23 27 8 16 2 25 32 ...]	[18 7 32 28 27 13 19 28 25 31]
9.	0.570	0.691	17.51	[15 39 9 14 9 7 15 22 39 27 ...]	[13 39 33 14 23 29 29 22 12 7]
10.	0.619	0.720	14.03	[18 29 0 20 39 12 39 6 30 38 ...]	[4 12 11 20 39 12 39 7 30 38..]
Avg.	0.586	0.711	17.36		

5 Further Optimization

In Section 3 we saw how to construct an optimization model for the cooperative assignment problem using the `coop`, `cweight` and `cando` arrays. It was also noted that the size of the model is quadratic in the number of users and number of tasks. The models were solved very fast for the data in our experiments, but the solution procedure can be speeded up even further. Our idea for optimization is based on partitioning the `coop` matrix into smaller matrices or partitions. This reduces the optimization problem into two smaller subproblems, which are solved separately and their results are combined. However, such decomposition is possible only when certain conditions are satisfied as we discuss shortly after an example.

Table 8: Partitioned Cooperation matrix

	Validate	Settle	Approve	Pay
Receive	1	1	0	0
Validate	-	1	0	0
Settle		-	1	0
Approve			-	1

Table 8 is an example cooperation matrix. This matrix can be divided now into two separate matrices or partitions P1 and P2 as shown in Table 9. A Partition P is defined by its rows and columns as: $P = \{P_{rows}, P_{cols}\}$. Thus, in Table 9, $P1 = \{(receive, validate), \{(validate, settle)\}$. Similarly, $P2 = \{(settle, approve), (approve, pay)\}$. We define valid partitions next.

Valid partitions: A matrix or a partition P can be divided into two smaller matrices or partitions P1 and P2 if the following conditions are satisfied:

- 1) $P1_{rows} \cup P2_{rows} = P_{rows}$ and $P1_{cols} \cup P2_{cols} = P_{cols}$
- 2) $|P1_{rows} \cap P2_{rows}| = 0$ and $|P1_{cols} \cap P2_{cols}| \leq 1$
- 3) $\forall j$ s.t. $(j \in P1_{cols}), \forall i$ s.t. $(i \in P) P(i,j) = 1 \rightarrow i \in p1_{rows}$
- 4) $\forall j$ s.t. $(j \in P2_{cols}), \forall i$ s.t. $(i \in P) P(i,j) = 1 \rightarrow i \in p2_{rows}$

Table 9: Decomposing the cooperation matrix into partitions

(a) P1: matrix for partition 1

	Validate	Settle
Receive	1	1
Validate	-	1

(a) P2: matrix for partition 2

	Approve	Pay
Settle	1	0
Approve	-	1

The two valid partitions for the coop matrix correspond to two smaller coop matrices and each one can be used to create a subproblem using the model of Section 3. Then each subproblem can be solved separately. Condition 1 above for a valid partition requires that all rows and columns in the original matrix be included in the new partitions for it. Moreover, by condition 2, the rows should not intersect while at most one column may intersect. Finally,

conditions 3 and 4 enforce the requirement that all '1' entries in a column in the original matrix should also appear in the partition in which the column appears.

In our running example of Table 9, there is no overlap between rows and only one overlapping task (*settle*) across the columns in the two partitions. The two partitions are used to formulate two subproblems SP1 and SP2. If both SP1 and SP2 find an optimal solution with the same value for *settle*, then it is the overall optimal solution. Otherwise, we can solve each subproblem again, while constraining the task *settle* to take another value from the set of feasible values for the task. In our example, the *settle* task can be assigned to Mike or Jim who are adjusters. Thus, we can alternately add "settle = Mike" and "settle = Jim" to the set of constraints for each optimization subproblem and find a solution. Then, the best overall solution can be found by combining and comparing the individual solutions.

```

1: Algorithm Optimize (coop[[[]], cweight[[[]], cando[[[]], constraints)
   Inputs: coop[[[]], cweight[[[]], cando[[[]] matrices, any additional constraints
   Outputs: Best value of the objective function, and (variable, value) pairs
2: (coop1, coop2) = Partition coop[[[]] into two valid partitions
3: If (valid partitions are not found)
4:   P1 = Formulate (coop[[[]], cweight[[[]], cando[[[]], constraints)
5:   Best.(obj, solution) = Solve(P1) // returns obj. function value and solution
6:   Return Best.(obj, solution)
7:   Exit
8: overlap_task = coop1_cols  $\cap$  coop2_cols
9: SP1.(obj, solution) = Optimize(coop1[[[]], cweight[[[]], cando[[[]], constraints)
10: SP2.(obj, solution) = Optimize (coop2[[[]], cweight[[[]], cando[[[]], constraints)
11: If (overlap_task =  $\emptyset$  OR value(overlap_task1) = value(overlap_task2))
12:   obj = SP1.obj + SP2.obj
13:   Best.solution = SP1.solution  $\cup$  SP2.solution
14: Else
15:   overlap = {set of possible value assignments for overlap_task}
16:   Foreach (val  $\in$  overlap)
17:     SP1[val].(obj, solution) =
       Optimize(coop1[[[]], cweight[[[]], cando[[[]], {"overlap_task=val"})
18:     SP2[val].(obj, solution) =
       Optimize(coop2[[[]], cweight[[[]], cando[[[]], {"overlap_task=val"})
19:     obj[val] = SP1[val].obj + SP2[val].obj

20:   val* = val s.t. obj[val*]  $\leq$  obj[val] for all val
21:   Best.obj = obj[val*]
22:   Best.solution = SP1[val*].solution  $\cup$  SP2[val*].solution
23: Return (Best.obj, Best.solution)

```

Figure 3: Optimization algorithm

The **Optimize** algorithm is shown in Figure 3. In this algorithm we first create two valid partitions of the coop matrix (line 2). If the partitions are not found then the problem is solved as a single optimization problem using a solver like CPLEX and the result is returned (lines 3-7). The **Formulate** function converts the input matrices and any additional constraint into the optimization model in the correct format for the solver. The result consists of the objective function value and (variable, value) tuples. Otherwise, the overlapping task between the two columns is identified (line 8). Next we solve two optimization problems *recursively* by calling the **Optimize** function itself (lines 9-10). If there is no overlapping task or if both solutions contain the same value for the overlapping task variable, then we can combine them by adding the objective function values and merging the sets of variables in the results to get the overall optimal (lines 11-13).

Problem SP1 gives a solution for the variables in $P1_{\text{rows}}$, while SP2 solves for variables in $P2_{\text{rows}}$. However, if the values for the task variable are different in the two solutions, then we identify all the possible values for the task variable (line 14) and recursively **Optimize** the subproblems by alternately adding a constraint assigning each value to the task variable (lines 16-19). The objectives of the solutions to the subproblems are added for each of these values and the one with the minimum value is taken as the optimal. Again the variables produced by the two subproblems are combined (lines 20-22).

If the coop matrix is sparse then this algorithm can speed up the processing by decomposing the problem into subproblems for the solver.

6 Further Extensions

In this section we consider some variants of the basic model. The first one includes cost in the model as a constraint or an objective. The second extension allows us to model varying

degrees of cooperation between agents instead of just 0-1 binary cooperation. Finally, we consider how to find an optimal assignment when multiple paths exist in the process.

6.1 Cost optimization

Our formulation in Section 3 has been presented in such a way so as to minimize incompatibility or equivalently to maximize compatibility. However, in many agent assignment scenarios [15,18, 19] the objective is to minimize cost. Given a user-task cost matrix, where $cost_{u,t}$ is the cost of agent u performing one instance of task t , the model can be modified easily. In such a case it is possible to incorporate cost into our model as an additional constraint such as:

$$\sum_{u,t} cost_{u,t} does_{u,t} < cost_{max}$$

Where,

$cost_{max}$ is the maximum allowable cost for the assignment.

Another alternative is to convert the objective function and add it as a constraint into a cost optimization model. Then the constraint is expressed as:

$$\sum fit_{u1,u2,t1,t2} * (1 - cweight_{u1,u2}) > 1 - compat_{min}$$

where $compat_{min}$ is a minimum desired compatibility threshold.

6.2 Varying degrees of required cooperation

Above the cooperation matrix only contained binary 0-1 entries for pairs of tasks, where a 0 indicated cooperation was not required between the agents performing two tasks, and a 1 indicated it was required. In general, varying degrees of cooperation may be required between agents of different pairs of tasks. For example, in the process of Figure 2, a high degree of cooperation (say, 0.9) may be necessary between the adjuster and the manager, and the need for cooperation between the manager and the accounts officer may be less (say, 0.3). This can be

captured by associating a continuous parameter between 0 and 1 to denote the strength of cooperation required between the performers of two tasks. Thus, the cooperation matrix would contain $coop_{t1,t2}$ entries that are values between 0 and 1, and not binary values. These values would be determined subjectively by somebody with knowledge about the process. Again, the objective function would also be modified as follows:

$$Minimize \sum_{u1,u2,t1,t2} coop_{t1,t2} p_{t1,t2} fit_{u1,u2,t1,t2} * (1 - cweight_{u1,u2})$$

The rest of the formulation would remain unchanged.

6.3 Multiple paths in a process

The process described in Figure 1 was a linear process. Now, consider a modified version of that process as shown in Figure 4. In this process, after the *validate* step there are two alternative branches. The upper branch is taken when the validation succeeds, i.e. the customer has a valid policy. However, if the validation fails, then the lower branch is taken. If the upper branch is taken then we obtain the same process as in Figure 1. However, if the lower branch is taken then it means then the claim is rejected outright without performing any further tasks. Thus, there are two paths in the modified process. In general, there may be many different such paths.

To handle this situation, we modify the objective function by introducing a new parameter $p_{t1,t2}$ for probability of cooperation between two tasks t1, t2. An example of a probability matrix is shown in Table 10 where the labels on the arcs give the transition probability between two tasks. In the revised objective function each term is multiplied by the corresponding probability that cooperation will be required between t1 and t2 as shown below:

$$Minimize \sum p_{t1,t2} fit_{u1,u2,t1,t2} * (1 - cweight_{u1,u2})$$

The constraints, however, remain the same as before.

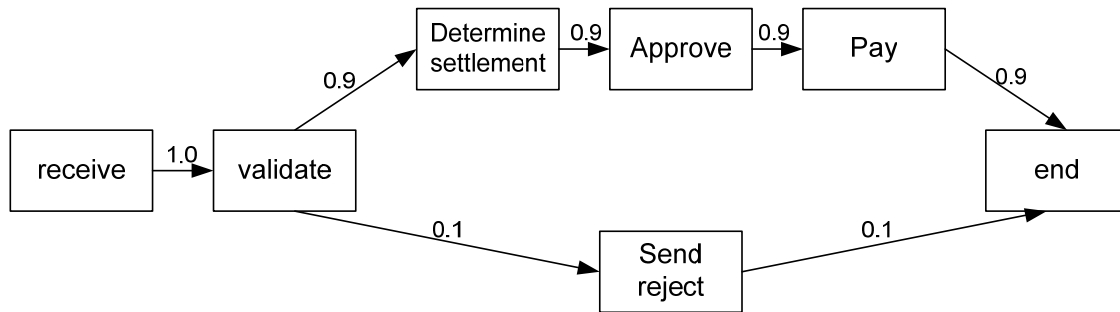


Figure 4: A revised process (labels on arcs show transition probabilities)

Table 10: A task-task transition probability matrix for the process in Figure 2

	Receive	Validate	Settle	Approve	Pay	Reject
Receive	-	1	0.9	0	0	0.1
Validate		-	0.9	0.9	0	0.1
Settle			-	0.9	0	0
Approve				-	0.9	0
Pay					-	0

7 Discussion and related work

We addressed the problem of work assignment where cooperation is important between agents operating as a team and showed how to model this problem in different ways, and also developed a faster solution procedure. By selecting a pair of agents who are more compatible with one another than another pair of agents, one can facilitate the smoother flow of work in an organization. It should also be noted that when a team of agents performs a workflow process, the assignment of agents to tasks can be done *dynamically* rather than making a *static* assignment at the start of the process instance. Thus, an initial assignment is made at the start, and after each successive task is completed, a new assignment can be made because the agent who was chosen

for the next task at the start of the workflow may not be available. Next we discuss how to compute the cooperation matrix.

7.1 Practical considerations – how to populate the cooperation matrix

In a practical setting, we can measure compatibility weights from empirical data based on logs. An example log is shown in Table 10. In this log there are 7 instances of an actual process. Each instance contains multiple tasks and each instance is performed by a different combination of agents. The last column gives the throughput time for the instance, and these times vary for different instances. The variations can result for different reasons, such as amount of work involved, efficiency of the agent, etc. If we compare the average throughput times for one combination of agents with those for another combination for multiple instances of the same process, then it is reasonable to argue that in large part the difference reflects the compatibility among agents.

Table 10: An example log showing agents and throughput time

Instance #	Task A	Task B	Task C	Task D	Throughput time
1	Joe	Betty	Sue	Mike	.		50
2	Lin	Betty	Sue	Mike			30
3	Joe	Betty	Sue	John			35
4	Lin	Betty	Sue	John			40
5	Dan	Betty	Sue	Mike	.		20
6	Lin	Jane	Sue	John			25
7	Lin	Bob	Sue	John			60
...							

Thus, if average throughput time was higher for those instances where Joe and Sue worked together as compared to when Joe and Mike did (all other tasks being done by the best set of individuals, i.e. those who lead to the shortest throughput), then the compatibility between Joe and Sue is higher than that between Joe and Mike. Then, a metric for *cweight* is:

$cweight(X, Y) =$

$$\left(\frac{\text{thruput}_{\max}(X, *) - \text{thruput}(X, Y)}{\text{thruput}_{\max}(X, *) - \text{thruput}_{\min}(X, *)} + \frac{\text{thruput}_{\max}(*, Y) - \text{thruput}(X, Y)}{\text{thruput}_{\max}(*, Y) - \text{thruput}_{\min}(*, Y)} \right) / 2$$

In this formula, we normalize the throughput gap when X works with Y versus the gap when X works with her most compatible worker. Similarly, we normalize the throughput benefit when Y works with X versus the benefit when Y works with her most compatible worker. As an example, we can apply the formula to calculate compatibility between Betty and Lin as:

$$cweight(Betty, Lin) = ((50-30)/(50-20) + (60-30)/(60-25))/2 = 0.762.$$

By analyzing the logs in this way, it is possible to populate the *cweight* matrix for compatibility between all pairs of agents.

7.2 Related work

When many agents or workers perform work collaboratively either on a team or on an instance of a running workflow, several factors can affect the overall performance. If the agents are distributed, their geographical location can play a role. In [12], based on an extensive empirical study it was shown that there is a positive effect on performance of workflow instances when agents are located geographically close together. This study has implications for assignment of work to distributed agents, and in relation to our work, it suggests that there may be a connection between geographical distribution of agents and their compatibility. Another factor that can affect performance is the delineation of task boundaries in breaking down a process or team project into pieces and assigning them to different individuals. In another study on factors that influence process redesign [14], it is shown that when tasks in a process are asymmetric and there is considerable customization of tasks to specific customer needs, process redesign can produce improvements. When the level of customization is high, effective communication becomes critical in ensuring a smooth handoff, and hence compatibility between

the agents carrying out the handoff is important. In the medical context also it has been shown that adopting appropriate handoff procedures can improve overall performance [10].

There is also related work on assignment of tasks to agents. In [8] assignments were made to agents based on criteria like suitability of the agent for the task, availability of the agent, urgency of the task, and degree of conformance between the agent and the constraints that apply to the task. However, this approach does not refer explicitly to teams although it can be extended into that context. Another approach for assigning work in emergency situations [11] is based on *threshold models* consisting of two components, threshold and stimulus. As stimulus associated with a task increases, even agents who have a high threshold for performing the task respond to it.

Yet another stream of research is directed towards assigning work based on optimization models in a variety of contexts such as: air crew scheduling [19], bus crew scheduling [15], assigning part-time employees and medical personnel [4, 18], etc. Most of these models are based on creating an IP formulation where the objective is to minimize cost subject to certain constraints. Finally, there has also been research conducted on teamwork models in the context agent systems [13,17]. Agents serve as proxies for robots and people, and teams of such agents are created in order to coordinate and communicate, and to take decisions in real world situations. In this context a notion of compatibility between agents is useful.

8 Conclusions

Effective business process teams are essential for organizational success. In this paper we have highlighted the importance of understanding compatibility when assigning work to teams or groups and shown how to model it. This issue has received very little attention in research literature. We developed a novel approach for building teams with high compatibility by

including agent-agent compatibility in our model. We have also shown how a heuristic compares with the optimal solution and found that the differences between the heuristic and optimal solutions can be quite large, on average 20% in our experiments and in the worst case more than 40%. Finally, we also described a faster solution procedure for our optimization model, and discussed various extensions to it to make the approach more versatile.

Acknowledgement: The author thanks Hajo Reijers and Minseok Song for helpful discussions during a visit to Eindhoven University of Technology.

References

1. Aalst, W.M.P. van der. The application of Petri nets to workflow management. *The Journal of Circuits, Systems and Computers*, 8, 1 (1998), 21–66.
2. Aalst, W.M.P. van der and K. van Hee. *Workflow Management: Models, Methods, and Systems*, MIT Press, March 2004.
3. Bonabeau, E., Dorigo, M., Theraulaz, G.: *Swarm Intelligence: From Natural to Artificial Systems*, Santa Fe Institute Studies in the Sciences of Complexity. Oxford University Press, Oxford (1999).
4. M. Grunow, H.-O. Günther, G. Yang. Development of a decision support model for scheduling clinical studies and assigning medical personnel. *Health care management science*. 2004 Nov;7(4): 305-17.
5. H, Haviv Y, Tuval A, et al., Improving Handoff Communications in Critical Care: Utilizing Simulation-Based Training Toward Process Improvement in Managing Patient Risk. *Berkenstadt, Chest*. 2008(Jul); 134(1):158–162.
6. ILOG. Ilog CPLEX software, Version 11.010, 2008.
7. Jablonski, S., and Bussler, C. *Workflow Management: Modeling Concepts, Architecture, and Implementation*. London: International Thomson Computer Press, 1996.
8. Kumar, A., van der Aalst, W.M.P., Verbeek, H.M.W. Dynamic Work Distribution. *Workflow Management Systems: How to Balance Quality and Performance*, *Journal of Management Information Systems* 18(3) (2002).
9. Kumar, A and Wang, J., "A framework for designing resource driven workflow systems," *The International Handbook on Business Process Management*, M. Rosemann and J. vom Brocke (eds.), Springer, 2009 (forthcoming).

10. K. Mistry, et al., Using Six Sigma Methodology to Improve Handoff Communication in High Risk Patients. Henriksen K, Battles JB, Keyes MA, Grady ML, editors. *Advances in patient safety: New directions and alternative approaches*. Vol. 1. Assessment. AHRQ Publication No. 08-0034-1. Rockville, MD: Agency for Healthcare Research and Quality; August 2008.
11. Reijers, H., A. Jansen-Vullers, M. H. Muehlen, M. z. Appl, W. Workflow Management Systems + Swarm Intelligence = Dynamic Task Assignment for Emergency Management Applications. *Proceedings BPM 2007, LCNS 4714*, pages 125-140, Springer-Verlag.
12. H.A. Reijers, M. Song, and B. Jeong. Analysis of a Collaborative Workflow Process with Distributed Actors. *Information System Frontiers*, 2008 (special issue on Collaborative Business Processes).
13. N. Schurr, et al. From STEAM to Machinetta: The evolution of a BDI teamwork model. *Cognition and Multiagent Interaction: From Cognitive Modeling to Social Simulation 2004*.
14. Seidmann, A. and A. Sundararajan, "The Effects of Asymmetry on Business Process Redesign," *International Journal of Production Economics* Vol. 50, (1997), PP 117-128.
15. Smith, Barbara M., "IMPACS - A bus crew scheduling system using integer programming", *Math Prog.* 42 (1988) 181-187.
16. Sun S. X., Zhao L, Nunamaker J and Sheng O (2006), Formulating the data flow perspective for business process management, *ISR*, 17 (4), pp. 374-391.
17. Tambe, M., "Teamwork in real-world, dynamic environments", *Proc. International Conf. on Multiagent Systems*, 1996.
18. Vakharia, A.J., Selim, H.S. and Husted, R.R., "Efficient scheduling of part-time employees", *Omega* 20 (1992) 201-213.
19. Wark, Pieter, Holt, John, Rönnqvist, Mikael and Ryan, David, "Generation of aircrew schedules using repeated matching", *Australian Society of Operations Research Bulletin* 15 (1996) 17-25.
20. Zur Mühlen, M.: *Organizational Management in Workflow Applications – Issues and Perspectives*. *Information Technology and Management* 5, 271–291 (2004).