Ant-Inspired Dynamic Task Allocation via Gossiping

Hsin-Hao Su
MIT

Lili Su
UIUC

Anna Dornhaus
University of Arizona

Nancy Lynch
MIT

May 7, 2017

Abstract

We study the distributed task allocation problem in multi-agent systems, where each agent selects a task in such a way that, collectively, they achieve a proper global task allocation. In this paper, inspired by ant colonies, we propose several scalable and efficient algorithms to *dynamically* allocate the agents *as the task demands vary.* Similar to ants, in our algorithms, each agent obtains sufficient information to make its local decision by gossiping with the other ants. Our algorithms vary in their advantages and disadvantages, with respect to (1) how fast they react to dynamic demands change, (2) how many agents need to switch tasks, (3) whether extra agents are needed, and (4) whether they are resilient to faults.

1 Introduction

In a multi-agent system, different tasks may need to be performed. The task allocation problem is to find an allocation of agents such that there are enough agents working on each task. This is often done in a distributed manner in many applications. For instance, drone package delivery for one city may consist of deliveries for several different regions [Koz17]. The drones may learn the demands in each region from a broadcasting ground control tower. The demands may vary from time to time. The drones are required to coordinate among themselves, without central control, to ensure that there are enough individuals working in each region.

The problem of task allocation also occurs in the ant world. In ant colonies, there are several different tasks (brood cares, foraging, nest maintenance, defense [Rob92]) which require different number of ants. Ant colonies generally do a good job of regulating the assignment of workers to tasks. In this work, we take inspiration from ants to develop several algorithms that are efficient and robust for the task allocation problem. Conversely, we hope our work can shed some light on questions about collective insect behavior.

To model the task allocation without centralized controllers, we consider randomized *gossip protocols* [DGH$^+$87] (sometimes known as *population protocols* [1] [AAD$^+$06]) as the underlying method of communication among the agents. In short, randomized gossip protocols consist of *rounds*. In each round, each agent chooses another agent uniformly at random to contact, and then the pair exchanges messages. Gossip-based protocols capture a common method of communication in biological systems. For example, in ant colonies, two ants communicate by touching each other with their antennae [Gor02]. The gossip protocol model captures the way they exchange information in a peer-to-peer manner. Not only are gossip-based protocols natural communication mechanisms in biological systems, the algorithms in such protocols are usually simple, easily scalable, and resilient to failures.

The Model We assume there are n agents and k tasks. Each agent a is associated with a unique identifier, $\text{ID}_a \leq \text{poly}(n)$, and a state $Q_a \in \{1, 2, \ldots, k\}$, which indicates the task that it is working on. The scenario proceeds in synchronized rounds. In the beginning of round t, each agent receives the demand signals $\vec{d}^{(t)} = (d_1^{(t)}, d_2^{(t)}, \ldots, d_k^{(t)})$ from the tasks, where $d_i^{(t)}$ indicates the demand of task i [2]. Note that the demands may change arbitrarily in every round. Each agent a chooses another agent a' uniformly at random and then they can exchange messages of $O(k \log n)$ bits (which fit the size of the input signals). Then, the agents can change their states. Then they proceed to the next round.

Cornejo et al. [CDLN14] and Radeva et al. [RDL$^+$] defined a model for the task allocation problem in ant colonies. In their work, when the ants receive the feedback from the environment, there could be information flow from one ant to another. In our model, *the information flow happens only through gossiping.*

Problem Formulation We formulate the task allocation problem similarly to [RDL$^+$] and [CDLN14] as follows. Let $A_i^{(t)}$ denote the set of agents working on task i for $1 \leq i \leq k$. Let $\vec{w}^{(t)} = (w_1^{(t)}, w_2^{(t)}, \ldots, w_k^{(t)})$ denote the number of workers working on the k tasks ($w_i = |A_i|$). We say the allocation at round t is a *proper allocation* if $w_i^{(t)} \geq d_i^{(t)}$ for all $i \in \{1, 2, \ldots, k\}$. For con-

[1] Though they are slightly different. E.g., population protocols usually have a more restrictive memory constraint.
[2] The demands should be thought as the work-rates required to keep the tasks satisfied.

Table 1: Comparisons of the Algorithms.

	Mv. and Fill	Tkn. Pass I	Tkn. Pass II	Ranking I	Ranking II
#Agents	D	$(1+\epsilon)D$	$(1+\epsilon)D$	$(1+\epsilon)D$	$(1+\epsilon)D$
Preproc. Time		$O(\frac{k}{\epsilon}\log n)$	$O(\frac{k}{\epsilon}\log n)$	$O(\frac{1}{\epsilon}\log^2 n)$	$O((\frac{k}{\epsilon})^2\log n)$
Realloc. Time	$O(\log^2 n)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
Switching Cost	OPT	$(k-1)\cdot$ OPT	OPT	$O(n)$	$O(k\log n)\cdot$ OPT (or $O(n)$)
Fault Tolerance		transient faults after preproc.		transient faults after preproc.	transient & (crash) no global clock

venience, assume that the total demand $D = \sum_{i=1}^{k} d_i$ is fixed. We can assume this without loss of generality, since we can let task k denote the dummy task for idle agents. We often omit the superscripts (t) to denote the quantities of the current round.

There are several objectives we consider for an algorithm. First, whenever the demands change, we hope that the allocation recovers to a proper one as soon as possible. The reallocation time is defined to be the number of rounds needed for the algorithm to find a proper allocation, after the demand stabilizes. Algorithms are allowed to have a preprocessing phase, so that the reallocation can be done faster after that. Second, when the demands change, we hope the number of task switches is as small as possible, since task changing may incur some overheads. We define the switching cost to be the number of agents who switched tasks until a proper allocation is achieved. When the demands change from $\vec{d}$ to $\vec{d}'$, it is clear that the switching cost is at least OPT $\overset{\text{def}}{=} |\vec{d}-\vec{d}'|_1/2$ (if the work exactly matches the demand for each task). Third, we study the number of agents needed for the algorithm. Clearly, all algorithms that behave correctly need to have at least D agents. However, the question is whether extra agents can help us in designing more efficient algorithms. Finally, we consider two types of faults: transient faults and crash faults. A transient fault means an agent temporary malfunctions but later recovers. For example, an agent might not receive the most recent demands for some reason (perhaps due to the propagation delay). We say an algorithm tolerates transient faults if the agents adapt to a proper allocation after all the agents recover from the faults. A crash fault is when an agent malfunctions permanently (and it will no longer be contacted by other agents). We say an algorithm tolerates crash faults if the agents adapt to a proper allocation after some of them crashed, as long as there are enough remaining agents.

2 Algorithms

In this work, we give three different types of algorithms for the task allocation problem. They are incomparable in the sense that no one dominates the other on all the objectives (see Table 1). Our first algorithm, the **move-and-fill algorithm**, is similar to the algorithm of Radeva et al. [RDL$^+$], where the excess ants working on over-satisfied tasks leave the tasks and switch to the unsatisfied tasks. We show that this can be done in $O(\log^2 n)$ rounds in our model w.h.p.[3] using the gossip-based counting and selection algorithms developed in [KDG03] . The main advantage of the algorithm over the other two is that the number of agents needed is exactly D. Moreover, the switching cost is optimal. The drawback of the algorithm is that whenever the demands change, the re-allocation time is $O(\log^2 n)$ rounds. If the demands change more frequent than $O(\log^2 n)$ rounds, the allocation will not be able to catch up to the demands. In reality, the demands may

[3]With high probability, which means with probability at least $1 - 1/\operatorname{poly}(n)$.

change very frequently due to both internal factors (consumable tasks where the demands decrease when they are done) and external factors (sudden changes in the environment).

The ant inspiration for the next two algorithms. Consider ant colonies, where ants receive the demand signals from the tasks. In reality, the signals can be the temperature or the stimulation of chemicals. Biologist have conjectured that different ants have different response thresholds to the signals [BTD98]. The question is whether such a design could help in task allocation. Consider the following simple example, where $n = k = 2$. Suppose that the first ant a_1 is more sensitive to the signal of task 1 than a_2. Then, when task 1 and task 2 have both 1 unit of demand, it is possible that a_1 goes to work on task 1 and a_2 goes to work on task 2. The main inspiration here is that if the ants have different responses to the signals, then they can take advantage of the difference to facilitate task allocation. Each ant can decide where to go based on the demand signals, independent of the other ants' actions. Therefore, the reallocation can be done very quickly.

Both our **token passing algorithm** and **ranking algorithm** are based on this idea. Both algorithms consist of a preprocessing phase, where each ant a computes a value X_a. After X_a is computed, they will allocate themselves according to X_a and the vector of demands, so that when the demands change, each ant can reallocate itself instantaneously. The drawback compared to the first algorithm is that they both need extra agents. After the X_a-values are computed, the allocation is done in a very simple way. In a high level sense, we divide the range of X_a-values into k disjoint intervals such that the length of i'th interval is proportional to the demand of task i (with additional slacks, see Algorithm 1). Every agent will go to the task whose interval contains its X_a-value. In general, we hope that the X_a-values of the agents are well-spread so that an interval of length proportional to d_i would contain d_i agents whose X_a-values lying in the interval.

Algorithm 1 allocate_task(a, X_a)

```
    Let $I_j = [\frac{D_{j-1}+\epsilon_{j-1}}{N}, \frac{D_j+\epsilon_j}{N})$, for $1 \leq j \leq k$, where $D_j = \sum_{i=1}^{j} d_j$,$\epsilon_j = j\lfloor \frac{\epsilon}{k} \cdot D \rfloor$, and $N = D + \epsilon_k$.
    Let $I_{j(a)}$ be the interval where $X_a$ is.
    Go to task $j(a)$.
```

In the **token passing algorithm**, each agent is assigned a unique token X_a from $\{1, 2, \ldots, n + \lfloor \frac{\epsilon}{k} \cdot D \rfloor\}$ in the preprocessing phase. This is done by using the loose renaming procedure developed by Giakkoupis et al. [GKW13]. When there are $(1 + \epsilon) \cdot D$ agents, the preprocessing phase takes $O(\frac{k}{\epsilon} \log n)$ rounds. After that, each agent can determine its role based on X_a and the demand vector in $O(1)$ rounds. There are two variants of the algorithms that reallocate in different ways when the demands change. The first is that every agent keeps the X_a-value the same and then reallocates according to that. In that algorithm, the switching cost is bounded by $(k - 1) \cdot \mathrm{OPT}$. In the second variant, the X_a-values are also reallocated. This achieves the optimal switching cost and the reassignments of X_a-values can be done instantaneously. However, unlike the first variant it does not tolerate transient faults after the preprocessing phase.

In the **ranking algorithm**, X_a is an estimate of the normalized rank (i.e. $\mathrm{rank}(a)/n$) of a, where $\mathrm{rank}(a)$ is the rank of a's ID over all the agents. In fact, the algorithm is more similar to ants' behavior. The ID of each agent can be thought as some features of the agents. In ant colonies, ants allocate the tasks based on their individual traits such as age [Rob92, TN04], body size [Wil80], genetic background [HSBB03]. For example, there are tendencies for older ants to go foraging and for younger ants to work on the tasks that are closer to the nests [TN04]. The ranking algorithms mimic this in the way that if the demands are fixed, then the allocation for every agent is determined by the relative position of its trait (assuming the traits are comparable).

We propose two different ways for estimating the normalized ranks. The first is a rounding-based algorithm that runs in $O(\frac{1}{\epsilon}\log^2 n)$ rounds while the second is a sampling-based algorithm runs in $O((\frac{k}{\epsilon})^2 \log n)$ rounds. The advantage of the first one is that the estimate does not depend on the execution of the algorithm and so that an agent always gets the same estimate. The advantage of the second algorithm is that it can tolerant both transient and crash faults. Moreover, each agent is allowed to keep its own clock, there is no global clock. The drawback is that the algorithms may have a fairly large switching cost (e.g., task switching can happen even when the demands are stabilized). However, for the second variant we may sacrifice the crash fault tolerance for a bounded switching cost of $O(k \log n) \cdot \mathrm{OPT}$.

3 A Lower Bound

When we fix X_a-values for the allocation, for the simple way of allocating the task described in Algorithm 1, the switching cost is capped at $(k-1) \cdot \mathrm{OPT}$. An interesting question is whether this is the best possible we can do. We may think of these algorithms as being inspired by the fixed-response thresholds model, because for each agent a, the reaction to a particular demand signal is always the same. We characterize such algorithms in the following definition.

Definition 3.1. A stable task allocation algorithm is where each agent a is associated with a function $f_a(d_1, d_2, \ldots, d_k)$ such that a goes to $f_a(d_1, \ldots, d_k)$ when the demand vector is $(d_1, d_2, \ldots, d_k)$.

Lemma 3.2. *Suppose that an algorithm is stable. Then there exists demands $\vec{d}$ and $\vec{d'}$ such that it takes at least $|d' - d|_1 = 2 \cdot \mathrm{OPT}$ switching cost when the demands change from $\vec{d}$ to $\vec{d'}$.*

Proof. Suppose there are 3 agents, a_1, a_2, a_3. Suppose to the contrary that $f_{a_1}, f_{a_2}, f_{a_3}$ are functions for a_1, a_2, and a_3 that achieve the optimal movements when there are 3 tasks with total demand 3. Suppose that the initial demand is $\vec{d_1} = (1, 1, 1)$. Without loss of generality, suppose that $(f_{a_1}(\vec{d_1}), f_{a_2}(\vec{d_1}), f_{a_3}(\vec{d_1})) = (1, 2, 3)$. When the demands change to $\vec{d_2} = (1, 2, 0)$, since we assume that the strategy achieves the optimal movement, we must have $(f_{a_1}(\vec{d_1}), f_{a_2}(\vec{d_1}), f_{a_3}(\vec{d_1})) = (1, 2, 2)$. If the demands again change to $\vec{d_3} = (0, 2, 1)$, then we must have $(f_{a_1}(\vec{d_1}), f_{a_2}(\vec{d_1}), f_{a_3}(\vec{d_1})) = (3, 2, 2)$ by the same reasoning. Finally, if the demands again change back to $\vec{d_1} = (1, 1, 1)$, then by the same reasoning, we have either $(f_{a_1}(\vec{d_1}), f_{a_2}(\vec{d_1}), f_{a_3}(\vec{d_1})) = (3, 2, 1)$ or $(f_{a_1}(\vec{d_1}), f_{a_2}(\vec{d_1}), f_{a_3}(\vec{d_1})) = (3, 1, 2)$, contradicting with the fact that $f_{a_1}, f_{a_2}, f_{a_3}$ are functions. □

4 Theoretical Problems Motivated by This Work

- **The ranking problem in the gossip model.** In our ranking algorithm, the preprocessing phase relies on the estimation of ranks for every agent. We have two algorithms for estimating the normalized rank up to $\pm\epsilon$ in $O(\frac{1}{\epsilon^2} \cdot \log n)$ and $O(\frac{1}{\epsilon} \cdot \log^2 n)$ rounds w.h.p. in the uniform gossip model. The question is whether one can do better, say in $O(\frac{1}{\epsilon} \cdot \log n)$ rounds.
- **The switching cost gap of stable algorithms.** We showed that stable algorithms cannot achieve the optimal switching cost (they must be at least 2-optimal). On the other hand, if all agents have their X_a-values properly assigned, then one can achieve a switching cost that is $(k-1)$-optimal. There is still a large gap between a factor of $(k-1)$ and a factor 2. Closing this gap is a very interesting open problem. The bounds are already tight when the number of tasks is three. Our partition scheme (Algorithm 1) shows that $2 \cdot \mathrm{OPT}$ is achievable, while our lower bound shows that this is the best possible. In fact, we have partial results showing that for $D \leq 7$ (see Appendix A), we can achieve a switching cost of $2 \cdot \mathrm{OPT}$. For $D > 7$, we could not generalize the pattern and therefore it is yet to be investigated.

References

[AAD$^+$06] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.

[BTD98] E. Bonabeau, G. Theraulaz, and J-L. Deneubourg. Fixed response thresholds and the regulation of division of labor in insect societies. *Bulletins of Mathematical Biology*, 60:753–807, 1998.

[CDLN14] Alejandro Cornejo, Anna R. Dornhaus, Nancy A. Lynch, and Radhika Nagpal. Task allocation in ant colonies. In *Proc. 28th Symposium on Distributed Computing (DISC)*, pages 46–60, 2014.

[DGH$^+$87] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proc. 6th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 1–12, 1987.

[GKW13] George Giakkoupis, Anne-Marie Kermarrec, and Philipp Woelfel. Gossip protocols for renaming and sorting. In *Proc. 27th Symposium on Distributed Computing (DISC)*, pages 194–208, 2013.

[Gor02] Deborah M. Gordon. The organization of work in social insect colonies. *Complexity*, 8(1):43–46, 2002.

[HSBB03] William O.H. Hughes, Seirian Sumner, Steven Van Borm, and Jacobus J. Boomsma. Worker caste polymorphism has a genetic basis in acromyrmex leafcutting ants. *Proceedings of the National Academy of Sciences*, 100(16):9394–9397, 2003.

[KDG03] David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *IEEE 44th Symposium on Foundations of Computer Science (FOCS)*, pages 482–491, 2003.

[Koz17] Stephan Kozub. Amazons new drone delivery plan includes package parachutes. *https://www.theverge.com/2017/2/14/14611242/amazon-drone-package-delivery-parachute-patent-prime-air*, 2017.

[RDL$^+$] Tsvetomira Radeva, Anna Dornhaus, Nancy Lynch, Radhika Nagpal, and Hsin-Hao Su. Costs of task allocation with local feedback: Effects of colony size and extra workers in social insects and other multi-agent systems. *submitted*. Preliminary version appeared as a brief announcement in *Proc. 28th Symposium on Distributed Computing (DISC)*, pages 657–658, 2014.

[Rob92] Gene E. Robinson. Regulation of division of labor in insect societies. *Annual Review of Entomology*, 37(1):637–665, 1992.

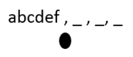
[TN04] Frederic Tripet and Peter Nonacs. Foraging for work and age-based polyethism: The roles of age and previous experience on task choice in ants. *Ethology*, 110(11):863–877, 2004.

[Wil80] Edward O. Wilson. Caste and division of labor in leaf-cutter ants (hymenoptera: Formicidae: Atta). *Behavioral Ecology and Sociobiology*, 7(2):157–165, 1980.

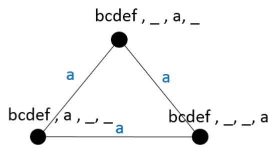
A 2-Optimal Switching Cost for Small Demands with 4 Tasks

We give a solution for achieving a 2-optimal switching cost for $n = D = 7$ and $k = 4$ by figures. For $D < 7$, the solution can be easily derived from them.

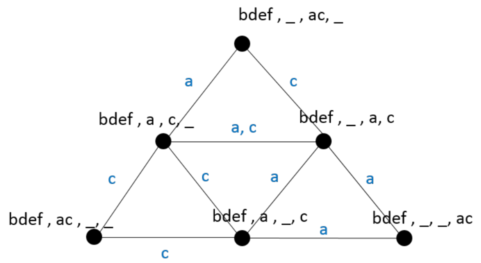
Figure 1: A solution for $k = 4$ and $D = 7$. Each node represents a demand vector. For example, the node in the first level represents $(7, 0, 0, 0)$. Therefore, the set of all demands (d_1, d_2, d_3, d_4) such that $d_1 + d_2 + d_3 + d_4 = 7$ form a 3D simplex. Two nodes are adjacent if the L_1 difference of the demands vectors they represents equal to 2. The labels on the edges denote the ants that switch tasks when the demands change from one endpoint to another (the edges across levels are omitted due to the difficulty of drawing). Since each adjacent move involves at most 2 agents, the solution is 2-optimal.



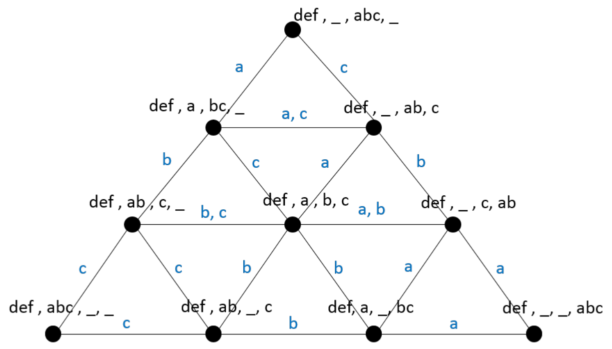
(a) Level 1



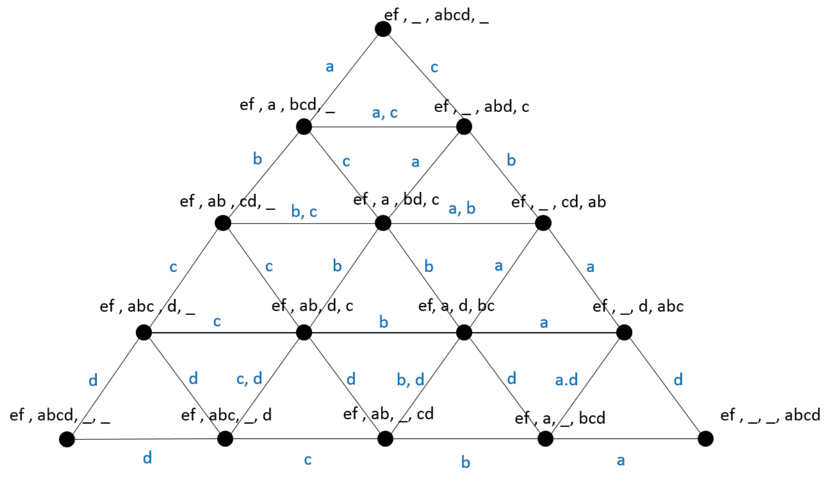
(b) Level 2



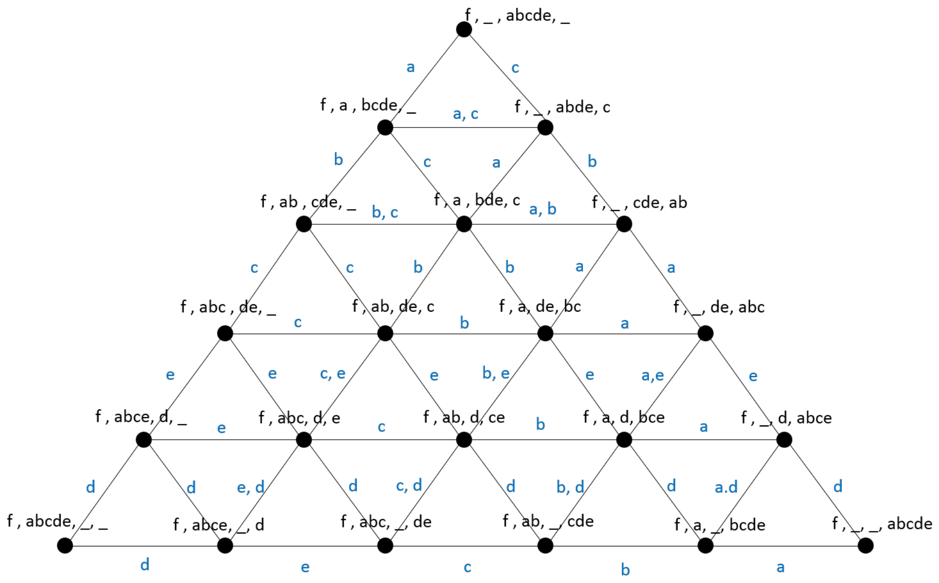
(c) Level 3



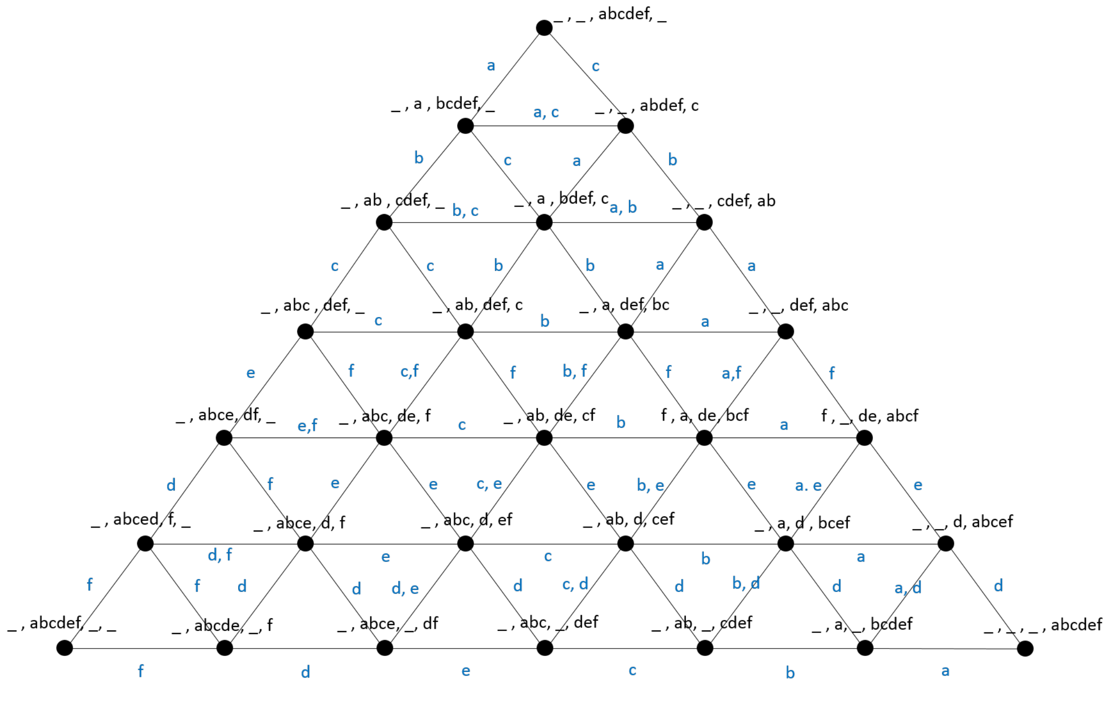
(d) Level 4



(e) Level 5



(f) Level 6



(g) Level 7