Chapter 5

Experiments

5.1 Settings

Originally, we aimed to reach 30-agents barrier. This raised the number of coalitions to 2^30 or 1 gigacoalitions. With the value-oriented algorithm we are going to use in the search process and, more importantly, the necessity to keep all the coalitions and their value, the original aim seems unreachable. We need at least 4 bytes ($8 \times 4 = 32$, each byte for representing the presence or absence of its respective agent) to represent one coalitions. Therefore, we need 4GB of memory to store coalitions alone. After that, we need another 4GB of memory to store coalition values (the implementation of the search algorithm was written in Java and it defines the coalition value as type `int` which requires 4 bytes of memory). In addition, we need extra 4BG for the sort process. As a result, we could not carry out the experiments up to 29 agents. Furthermore, the nature of breadth-first search approach deployed in our algorithm, some settings were not completely experimented because they required even more memory.

We experimented our algorithm by mean of simulation, i.e. each agent run on a single computer. Due to the limited budget we were allocated, we could only manage to acquire a single computer, equipped with 8 GB of memory, and a single Intel (R) i7-2600K 8-core @3.4GHz CPU. Since we want to minimize communication among agents, we assume the extreme case where the agents could not or would exchange the information about the progress of their search only once at the end of their individual search, which are logically indifferent.

As in the literature, we consider the data distribution of coalition values in two dimensions, i.e. the horizontal and the vertical views. In the horizontal view, if we plot the highest coalition value of each cardinality on a graph,

whose x-axis represents the cardinality and y-axis represents the coalition values, and draw a line from each of them from small to large coalitions, we will have 6 trends, as previously discussed in the review section, i.e. i) STD, ii)IND, iii) DCD, iv) CCD, v) CVD, and vi) RDD (for which we cannot really define a simple trend). In each of these trends, we have the vertical view, where we consider the distribution of coalition values in each cardinality. In this view, we have 3 other normal distributions, i.e. *i*)**f1.0** where the mean of coalition values are close to the top end of the coalition value range in that cardinality, *ii*)**f0.5** where the mean of coalition values are in the middle of the coalition value range in that cardinality, and *iii*)**f0.1** where the mean of coalition values are in the bottom of the coalition value range in that cardinality. Therefore, we have now 18 settings. In addition to this, we also implemented the 2 classical distribution patterns used in old fashioned experiments, i.e. the normal distribution (NMD) and the uniform distribution (UNI) for the sake of the completeness. All together we have 20 settings.

We carried out our experiments only on the extreme cases, i.e. 28 and 29 agents, which superseded previous experiments in the area. However, due to limited resources as discussed before, some settings for 29 agents failed to complete the experiments. In each of these settings, we run our algorithm for 100 times. We kept track of what was going on significantly, i.e. when the new solution was found in each run. Originally, we set the time out for each run to 10 minutes. In the worst case, this would take 10 x 100 x 20 x 2 = 40,000 minutes or 6667 hours or 278 days or approximately 9 consecutive months without any failures. After some time, we realize it was very hard to achieve because there were some errors in memory management inside the computer (we cannot really blame the automatic garbage collection mechanism of JVM because memory management is always error prone, regardless of the programming language.) We then reduce the time by half, i.e. 5 minutes. This is still time consuming, i.e. in the worst case, the time required for completing the experiment would be 4 months. Given the fact that most of the time during the course of the research was spent on the two main parts: the partition of the search space and improvement of the search method, we did not have that much time left.

After several times of changing partitioning methods and unsuccessful experiments, we finally decide to set the time out to just 1 minute for each run. This would require approximately a month of consecutive runs in the worst case. At first, without careful consideration, this setting may sound unreliable. However, given the fact that the performance of the search algorithm is measured in milliseconds and the clock speed of the CPU is 3.7 GHz, or a million times in a milliseconds, which is relatively fast, the time

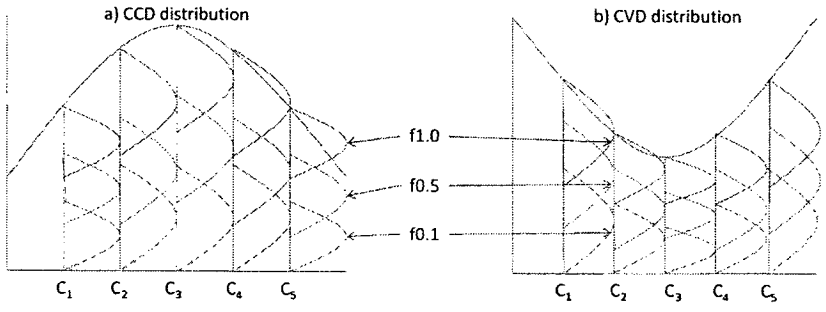

Figure 5.1: Trends of coalition values in various distribution form.

out of 1 minute is sound. More importantly, as the results turned out, we have satisfiable performance and that proved the efficiency of the algorithm as well as the reliability and the justification of our experiments.

The data, coalition values and their distribution patterns, for each run are generated in a specific way. The setting, i.e. the number of agents, the maximal value of the coalition, the trend of the horizontal view of the data distribution, the minimal value of the coalition, the deviation of the coalition values from the mean, the characteristic of the mean of the coalition values in each cardinality, the global seed, and the seed for generating coalition values for each cardinality. These data will be recorded in a file, whose name is in the form $s - n99t9f9.9 * .dat$, where $n99$ indicates the number of agents, e.g. $n28$, $n29$, etc., $t9$ indicates the type of trend of horizontal view of coalition values, e.g. $t0$ for STD, $t1$ for IND, etc., $f9.9$ indicates the mean of the coalition values in each cardinality, e.g. $f1.0$ for those whose mean are close to the top end of the range of the coalition values, $f0.5$ for those whose mean are in the middle of the range, etc., and * represents the time when all these data were generated.

There were 100 files, each of which is for each run, for each setting. For each setting, we created a script file which would ran the java class, implementing our algorithm, and pass as the input for each run the name of a setting file and the time out. Originally, we had set the run for 5 x 8 x 3 = 120 settings. However, the limited resource on our computer did not allow us to complete the task as we have mentioned before.

5.2 Results

During each run in the experiments, there is a log file in which we kept track of the progress of the algorithm. This log file simply reuse the name of its

respective setting file and add the ".log" extension to the name. Every time the new solution has been found, the the elapsed time and the new coalition structure value, as well as other details including the coalition structure and its respective coalition values, will be recorded in to this file. Before termination, either by the optimal value has been found (via means of branch and bound proof) or time out, the last solution would be recorded.

After completed all the run, the last two records of each run for each setting were accumulated. The last record tells the total execution time of that run and the final solution. We assume the final solution is the (near) optimal coalition structure. The second last record tells time the time it took to reach optimality and the (near) optimal coalition structure, which is the same as that of the last record. For each run, we then find the difference between the total execution time and the time it took to reach optimality. We also find the difference between the values of the last and second last solutions. The most important information is the elapsed time it took to reach optimality, while the second most important information is the difference between the total execution time and the elapsed time it took to reach optimality. The former tells us how quick the algorithm could reach optimality while the latter tells us how quick the algorithm could prove that it had reached optimality and could terminate. Note that the algorithm could not be sure whether it had reached optimality unless it had proved that there were no chances that it could improve the solution. Another interesting dimension is how the algorithm progress before it could reach optimality. This issue is important when the algorithm took longer time to reach optimality. If the progress curve is convex then it is good because the algorithm could reach optimality quickly.

For each setting, we find the average elapsed time, the average terminated time, the difference between the average elapsed time and the average terminated time, and the average of the difference between the final solution and the second last solution. We did not compare our result against any other algorithm simply because we work in a more specific environment where other works are not applicable. We present the conclusive results in Table 5.1 and Table 5.2.

Distribution	Settings	Average elapsed time	Average terminated time	The difference between the average elapsed time and the average terminated time	The average of the difference between the final solution and the second last solution
STD0.1	s-n28t0f0.1	15114	55740	40626	0
STD0.5	s-n28t0f0.5	141	60004	59863	0
STD1.0	s-n28t0f1.0	72	60005	59933	0
IND0.1	s-n28t1f0.1	640	11629	10989	0
IND0.5	s-n28t1f0.5	1317	20142	18825	0
IND1.0	s-n28t1f1.0	577	43807	43230	0
DCD0.1	s-n28t2f0.1	13789	58642	44853	0
DCD0.5	s-n28t2f0.5	107	60005	59898	0
DCD1.0	s-n28t2f1.0	72	60005	59933	0
CCD0.1	s-n28t3f0.1	9893	57634	47741	0
CCD0.5	s-n28t3f0.5	70	60005	59935	0
CCD1.0	s-n28t3f1.0	72	60004	59932	0
CVD0.1	s-n28t4f0.1	2719	11578	8859	0
CVD0.5	s-n28t4f0.5	4621	14761	10140	0
CVD1.0	s-n28t4f1.0	372	40163	39791	0
RDD	s-n28t5f0.1	8979	36986	28007	0
NMD	s-n28t6f0.1	13505	57622	44117	0
UNI	s-n28t7f0.1	2519	42119	39600	0

Table 5.1: This table presents the empirical results for 28 agents.

Distribution	Settings	Average elapsed time	Average terminated time	The difference between the average elapsed time and the average terminated time	The average of the difference between the final solution and the second last solution
STD0.1	s-n29t0f0.1	16921	59542	42621	0
STD.5	s-n29t0f0.5	182	60008	59826	0
STD1.0	s-n29t0f1.0	130	60008	59878	0
IND0.1	s-n29t1f0.1	2200	36226	34026	0
IND0.5	s-n29t1f0.5	2414	45961	43547	0
IND1.0	s-n29t1f1.0	1468	53129	51661	0
DCD0.1	s-n29t2f0.1	13436	59116	45680	0
DCD0.5	s-n29t2f0.5	127	60008	59881	0
DCD1.0	s-n29t2f1.0	126	60009	59883	0
CCD0.1	s-n29t3f0.1	14210	59700	45490	0
CCD0.5	s-n29t3f0.5	120	60008	59888	0
CCD1.0	s-n29t3f1.0	127	60008	59881	0
CVD0.1	s-n29t4f0.1	17342	38814	21472	0
CVD0.5	s-n29t4f0.5	16692	60007	43315	0
CVD1.0	s-n29t4f1.0	133	60009	59876	0
RDD	s-n29t5f0.1	11974	53082	41108	0
NMD	s-n29t6f0.1	14408	59364	44956	0
UNI	s-n29t7f0.1	5496	53586	48090	0

Table 5.2: This table presents empirical results for 29 agents.

5.3 Discussion

From the data presented in Table 5.2, 5.2, we will discuss the results as following. In STD distribution, the termination times of all settings f0.1 in both 28 and 29 agents are slightly shorter than that of f0.5 and f1.0, which are about 1 minute. However, the time it takes to reach optimality,which we shall refer to as optimality time, hereon, are much shorter but with a little variation. In f0.1 of both 28 and 29 agents, the algorithm took about a quarter of a minute to reach optimality. However, in f0.5 and f1.0, the algorithm took much less time, i.e. less than two tenth of a second in f0.5 and about a tenth of a second in f1.0. This tell us that the it is hard for the algorithm to prune the search space and took more time to terminate.

In IND distribution, the termination times of all settings are shorter than that of IND with an ascending trend from f0.1 towards f1.0 in both 28 and 29 agents. With regards to the time to reach optimality, the algorithm took less time in t0.1 and t1.0 than that of t0.5, both in 28 and 29 agents, i.e., around half a second and around 2 seconds against around one and a half seconds and around two and a half seconds. However, the optimal time of t1.0 in both 28 and 29 agents are slightly less than that of t0.1. Therefore, in IND the algorithm can prune slightly better than in STD.

In DCD distribution, the termination times are very similar to that of STD both in vertical view of data distribution and in 28 and 29 agents. Furthermore, the times the algorithm took to reach optimality also have similar pattern to that of STD in all vertical view of data distribution of both 28 and 29 agents. More precisely, the algorithm took just less than a quarter of a minute to reach optimality in f0.1 of 28 and 29 agents. On the other hand, the algorithm took just around a tenth of a second to reach optimality in f0.5 and f1.0 of both 28 and 29 agents. This tells us that the behavior of the algorithm in STD and DCD is similar.

In CCD distribution, the termination times are again similar to that of STD and DCD in all settings. The times the algorithm took to reach optimality are also similar to the pattern of STD and DCD. However, there are some differences in details. The optimality time of the algorithm in f0.1 is less ten seconds in 28 agents, while it took around a quarter of a minute to reach optimality in 29 agents. On the other hand, the algorithm took just less than a tenth of a second to reach optimality in f0.5 and f1.0 of both 28 and 29 agents, while it took slightly more than that to reach optimality in f0.1 Apparently, the behavior of the algorithm in CCD is similar to its behavior in both STD and DCD.

In CVD distribution, the behavior with respect to the termination time of the algorithm has slightly different pattern to that of STD, DCD and CVD.

More precisely, the termination time of f0.1 and f0.5 in 28 agents are about a quarter of a minute but it goes up to about three quarters of a minute in f1.0 of 28 agents. The termination time of the algorithm in f0.1 of 29 agents goes above half a minute and it goes up to a minute in f0.5 and f1.0 in 29 agents. With respect to optimal time, the behavior of the algorithm in CVD is similar to that of the algorithm in IND, i.e. the algorithm took less time in f1.0 of both 28 and 29 agents, around a third of a second and a seventh of a second respectively. In f0.5 and f0.1, the algorithm took less than three and five seconds in 28 agent. However, the figures go strangely high in 29 agents, i.e. around 17 seconds.

In RDD, NMD and UNI, the behavior of the algorithm follow a consistent pattern in all f0.1, f0.5 and f1.0. With respect to termination time, the algorithm took around 37, 58 and 42 seconds in 28 agents. In 29 agents, the algorithm took around 53, 59 and 54 seconds to terminate. This implies that the algorithm can prune the search space in these settings. With respect to the optimal time, we can find some consistent behavior of the algorithm in both 28 and 29 agents.. The algorithm spent most time in NMD, second most time in RDD and the least time in UNI, i.e., around 9 and 12 seconds, 14 seconds and two and a half and four and a half respectively. In general, these RDD, NMD and UNI share some similarity in their data distribution.