Forming Productive Student Groups Using a Massively Parallel Brute-Force Algorithm

Tyson R. Henry

Abstract—The ability to work in productive groups is critical for success in the software industry, thus computer science students must be given the opportunity to work in productive groups. However, forming productive groups is difficult. A promising new way to form student groups is to use a web application to collect information from individual students and then use an algorithm to form the groups. This paper provides an overview of group formation, presents existing group formation work, explains the complexity of the group formation problem, reports on a set of experiments that provide an empirical description of the data topology of group formation and subsequently illustrates the shortcomings of existing algorithms. Specifically, the solution space is very sparse and probabilistic algorithms are unlikely to find a good solution. Finally, a massively parallel limited brute-force group formation algorithm is presented.

Index Terms—student groups, student teams, group formation, team formation, groupware.

I. Introduction

Students need productive group experiences for two reasons. First, potential employers usually ask students about their group experiences. Students who relate positive and productive experiences are more likely to be hired. Second, once employed in the software industry, success often depends on one's ability to be a productive member of a software development group. Positive group experiences from school can help students learn how to be productive in industry settings and subsequently help them succeed early in their career. While the need for productive groups is clear, no silver bullet solution exists for how to form productive groups.

There are three traditional approaches for forming student groups: random composition, self selection, and the instructor manually creating groups. Each has significant shortcomings: random can lead to very unbalanced groups and is unlikely to produce effective groups, self selection discriminates against less connected students, and the large number of possible groupings makes manual grouping difficult and unlikely to produce one of the best possible groupings. These shortcomings have led to a growing body of research into algorithmic group formation.

Algorithmic group formation uses information about individual students (i.e. goals, interests, grades, availability for meetings, etc.) to form the groups. The first step is to create a survey and have all students complete it. The

Manuscript received June 25, 2013; revised August 16, 2013. This work was supported in part by a gift from the NVIDIA Corporation.

Tyson R. Henry is with the Computer Science Department, California State University, Chico, CA 95926-0410 USA; 530-898-5709; e-mail: trhenry@csuchico.edu.

or weight of each possible group. An integer weight is information students provide is used to evaluate the strength calculated based on information provided by students. For example, a group that is available to meet ten times during the week would have a larger weight than a group that can only meet three times (see [1] for a thorough description of how the weights are calculated). The task of the group forming algorithm is to find the set of groups (or grouping) with the largest total weight. For example, if a grouping contains $group_i$, $group_j$, and $group_k$, the weight of the grouping is $weight(group_i) + weight(group_j) + weight(group_k)$.

Existing group formation systems use a web-based front end to gather information from students and one or more heuristic algorithms to generate the groupings. While these systems have improved on the traditional methods, they have not adequately addressed the complexity of the problem or considered the topology of the data space and thus are unlikely to find optimal groupings.

This paper provides an overview of existing algorithmic formation, describes the complexity of the problem, reports on some experiments that demonstrate the inadequacy of existing algorithms, and describes a massively parallel group formation algorithm. A web-based system that uses the presented algorithm for forming groups is described in [2].

II. Related Work

Many group formation systems are described in the literature [3]-[7]. The most documented and the only system available on the web is Team-Maker [3]. A large effort (called CATME) is currently underway to provide a thorough assessment of Team-Maker [5]. It provides a web-based system any instructor can use to create groups and is using the data from actual courses to evaluate their group formation process.

The CATME research is well formed and is likely to produce valid results. However, it does not directly analyze the grouping algorithm. It assumes the groupings produced by its algorithm are near optimal. The analysis described here demonstrates that it is unlikely the CATME algorithm finds near optimal results.

The literature contains descriptions of many different approaches to formulating groups: genetic algorithms [5] , a hybrid grouping genetic algorithm approach [8], fuzzy-genetic decision support system [9], evolutionary algorithms [10], ant colony optimization [11], agent-organized networks [12], fuzzy optimization approach [13], student performance [14], analytical hierarchy process [7], clustering [15], probabilistic model checking [16], graph

ISBN: 978-988-19252-3-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

analysis [17], and integer programming [18]. This list shows the breadth of research on algorithm based group formation. However, there has been only minimal analysis of the results. Many of the listed researchers compare the groups composed by their algorithms to randomly composed groups or groups created manually by an instructor. Other researchers evaluate their algorithms by comparing the results to results generated by the same algorithm without some portion of the logic. There is clearly a need for a better understanding of the grouping data and a means of evaluating algorithms.

III. Grouping Problem Complexity

The number of possible groups of size **g** that can be constructed from a class of **n** students is *n choose g* or $\binom{n}{g}$. This is equivalent to $\left(\frac{n!}{g! \cdot (n-g)!}\right)$. For a typical class, this set is small enough that it can be constructed quickly. For example, a class of 36 students partitioned into groups of 6 students can be partitioned into 1,947,792 distinct groups. A modern desktop computer can calculate these groups and their weights in a matter of seconds. The computational road block arises from the task of selecting the best set of groups (or best grouping). The number of possible ways to group all the groups into groupings is

$$\frac{\left(\frac{n!}{g! \cdot (n-g)!}\right)!}{\left(\frac{n}{g}\right)! \cdot \left(\frac{n!}{g! \cdot (n-g)!} - \frac{n}{g}\right)!}$$

For a class of 36 students partitioned into groups of 6 students there are 2.67×10^{24} possible groupings. This problem is intractable; it is not feasible to consider all the possible groupings. In order to find good groupings, a heuristic algorithm that considers only a fraction of possible groupings must be used.

IV. Data Topology Experiments

Existing group formation algorithms make two assumptions, (1) there are many good groupings and thus a heuristic algorithm started at a random location is likely to be near a good solution, and (2) the best groups live at the top of well formed hills. These claims were investigated by a series of experiments to explore the data topology.

A software engineering class of 25 students took an extensive survey that included questions about previous grades, the types of projects they are interested in working on, when they are available to meet, and their preferences for working with each of their classmates. This data was used to assign a weight to the 53,130 possible groups of five students.

In order to estimate the likelihood that a given heuristic algorithm would find an optimal solution, an accurate model of the data is required. It would be ideal to explore the entire data set (all possible groupings) but the data is too large (6.23×10^{14} for this example). Thus one or more assumptions about the data must be made in order to limit the number of possible groupings to be considered.

For these experiments, it was assumed that the best groupings would contain the best groups. In other words, groups with the highest weights will probably be the ones in the best groupings (the groupings with the highest weight). Using this assumption, the following algorithm was developed to search for optimal solutions:

- Sort all groups from largest weight to smallest smallest.
- Consider the first N best groups (which are now at the front of all groups).
- For every $group_i$ where $i < N$, recursively consider all possible groupings that contain $group_i$ and $groups_k$ where $k > i$.
- The search is terminated when it becomes mathematically impossible to improve on the current best grouping (since the groups are ordered from best to worst, it is possible to determine when no better solutions exist).

The result of this algorithm is the best grouping for each of the N best groups (since all possible groupings are considered, it is certain this is the best grouping for the given group). However, this set of N groupings may not contain the best overall grouping. This set of groupings will contain the best overall grouping only if the assumption that the best groupings contain the best groups is true.

The algorithm was run on a 2000 GFLOPS computer (one with two 448 core NVIDIA Tesla C2050 GPU computing processors) for about 40 days. Ten threads were run on each core resulting in 8960 threads. Each thread found the best grouping for two groups. Thus the program considered the best groupings for the best 17,952 groups (34% of total 53,130 groups). The results of this experiment are shown in Figure 1.

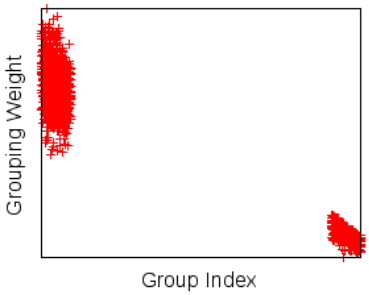

Fig 1: Best 17,952 Groups (with pruning)

For each processor, the current best grouping was stored. When considering a new starting group, if it became mathematically impossible to improve on the best current grouping (because all the remaining groups had weights too low to produce a better grouping), the search was pruned. The middle of Figure 1 is empty because each of these groups lead to a groupings with a lower weight than the

ISBN: 978-988-19252-3-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

thread's current best grouping (best groupings are shown on the left). The second group of plots were generated by the second set of threads. These threads started their searches by starting with groups later in the array. These threads also pruned the majority of their searches. The results shown in Figure 1 validate the assumption that the best groupings will contain the groups with the highest weights.

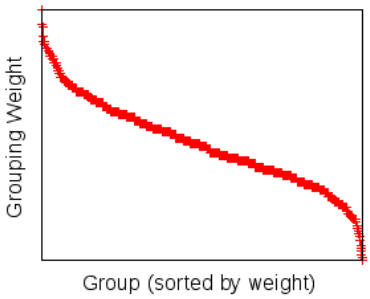

Fig 2: Best 896 Groups
(no pruning)

In order to expand on the data shown in Figure 1, a second experiment was performed starting with the 896 groups that composed the best groupings found in the first experiment (896 was chosen to facilitate dividing the task among the 896 processors). For each group, the best grouping was found. The sorted results are shown in Figure 2.

The first experiment illustrated that it is unlikely optimal groupings are in the data space not explored. The steep decline in Figure 2, shows that it is unlikely that even good solutions (those with weights less than half of the best groupings) are in the unexplored data space.

The net results of both experiments is that there are very few good solutions (those with weights greater than 50% of the best groupings). Thus a heuristic algorithm must use more than chance to guide itself toward a good solution.

Existing group formation algorithms (e.g. [3]-[7]) rely on the data space containing many good solutions (they typically consider only one in 100,000,000,000 possible groupings). These algorithms rely on forms of hill-climbing that in turn relies on the data containing well formed hills. A third experiment was performed to determine if the best groupings are at the top of well formed hills.

For the 896 best groupings found above, all 250 possible single swap permutations were considered (a student from one group was swapped with a student from another group; this resulted in 250 possible new groupings). Then for each of the groupings that resulted from the swaps, all 250 possible single swap permutations were considered, and so on. After four steps of single swaps, it is possible to reach as many as $896 * 250^4 = 3.4 \times 10^{12}$ groupings. At each step of the way, if the swap did not produce a lower weight grouping (a grouping downhill from the last step), or if a hill climbing algorithm would not choose to move towards the last step (because it was not the best choice), the grouping was pruned.

Of the 896 best groupings, only 14 were *uphill* from neighbors 4 steps away. For these 14 groupings, on average only 2.7% of all their neighbors 4 steps away were strictly *downhill* from them. This result compounds the effect of the sparse data. Not only is it very unlikely that a random algorithm could get close to an optimal solution, it is unlikely (2.7% chance) that the algorithm would move towards the optimal solution even if it landed close to it.

V. Limited Brute-Force Parallel Algorithm

The algorithm used for the experiments reported above combined with more aggressive pruning results in an algorithm that can find good groups in a reasonable time (a few minutes). This section provides the implementation details of this algorithm, and discusses the pruning approaches.

Each group is represented using an array of bits. For example, if the class contains 64 students, 64 bits are used to represent a group. If student **n** is in group **g**, bit **n** of group **g**'s bit array is set to 1. If student **n** is not in the group, bit **n** is set to 0. The current system limits class size to 64 which allows for a single long int to represent each group and thus comparisons can be made in a single CPU cycle.

A valid grouping is a set of groups that contain all the students in the class. Since each student is represented by a unique bit, when the bitwise-or operator is applied to all the groups in a grouping, the result must be a bit array in which all the elements are 1. For example, consider a class of four students to be grouped in groups of 2. The possible groups are:

0011 0101 1010 1001 0110 1100

The first group (0011) contains $student_0$ and $student_1$ (bits are numbered right to left).

These groups can be combined into the following groupings:

0011 and 1100
0101 and 1010
1001 and 0110

When the groups are combined using bitwise-or, the result is a bit array that contains all the students:

0011 | 1100 = 1111
0101 | 1010 = 1111
1001 | 0110 = 1111

The first step of the algorithm is to generate all possible groupings. This is equivalent to all possible bit arrays that represent a valid group (that is, all bit arrays of length equal to the class size and with **m** non-zero bits where **m** is the group size). This step uses recursion with backtracking, similarly in fashion to searching for solutions in the 8-queens problem.

Once all the groups have been enumerated, a weight is calculated for each group. The method for calculating the group weights is similar to that described in [1]. The groups are then sorted from highest weight to lowest weight.

The next step is to eliminate groups that are undesirable.

ISBN: 978-988-19252-3-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

For example, research shows that in male dominated classes, women tend to do better when in a group with at least one other woman [19]. Thus groups that contain a single woman can be removed from the set of groups. Another example would be to prune groups that have low social sensitivity [20]. Such semantic pruning can be controlled by the group formation website.

The set of groups must then be pruned along two metrics. The first is the number of groups to consider. Since the groups are sorted highest weight to lowest weight, it is likely that the groups that form the best groupings are those at the front of the list. In practice, considering 2,000 – 5,000 *seed groups* can yield several strong groupings for class sizes around 30.

The second pruning metric is which groups will be part of the seed group set. The group searching algorithm (described below) is seeded with a set of groups likely to be in good groupings (e.g., groups with high weights). In order to make the algorithm fair for all students, a set of seed groups for each student is found. Thus the total number of seed groups is distributed among the students. For example, if the seeding set is 3000 groups and there are 30 students, 100 seed groups will be found for each student.

At this point the best grouping is found for each seed group. Specifically, the grouping with the highest weight that contains the seed group g_i and other groups with an index greater than i. This is sufficient because if there were a better grouping for $group_i$ that contained a group with index < i, that grouping would be found when considering one of the groups with an index < i as the seed group.

Recall that all the groups are in an array. g_i is the group at index i. This algorithm considers all groups with an index > i. When forming a grouping two things are stored: the indexes of the groups in the grouping (stored as an array of integers) and the bitwise-or of all the groups (stored as a long integer) that represents the students in the current grouping. The algorithm uses recursion with back-tracking similar to typical solution to the 8-queens problem.

It starts with the seed group index in the array of indexes and the seed group as the current grouping members (the bitwise-or of all groups in the grouping). It then considers group g_{i+1} by performing a bitwise-and with the new group and the current grouping members. If the result of this bitwise-and is zero (which happens when none of the members of $group_{i+1}$ are already in the grouping), this group is added to the grouping (its index is added to the array of indexes and it is added to the current grouping members using bitwise-or).

Before a new group is added to the current grouping, the current grouping is saved as a potential backtracking point. In other words, the algorithm will eventually backtrack to the point where $group_{i+1}$ was added. When it backtracks to this point, $group_{i+1}$ will be skipped and $group_{i+2}$ will be considered.

The algorithm continues adding groups to the current grouping until it either completes a grouping (i.e. contains a set of groups that includes all students in the class) or reaches the end of the groups without finding a complete grouping. If a valid grouping was found, it compares the weight of the grouping with the best grouping found so far (the weight of a grouping is the sum of the weights of the groups in the grouping). If the weight is higher, this new grouping becomes the current best grouping.

At this point the algorithm backtracks to the point where the last group was added. It skips this group and continues. This method exhaustively considers all possible groupings containing $group_i$ and groups with index > i.

Consider the situation where the target grouping size is six groups and the current grouping contains three groups and has a weight of 300. If the current group being considered has a weight of 50, the three groups that will need to be added to this grouping will all have weights less than or equal to 50. Thus the the best possible weight for the current grouping is 450 (300 + 3 * 50). If the best grouping for the current seed group is greater than 450, this search can be terminated (pruned) with certainty that a better grouping cannot be found.

The algorithm is executed using NVIDIA graphical processing units (GPUs). (The two Tesla K20 GPUs used for this research were donated by the NVIDIA Corporation.) The algorithm is distributed among the processing cores. Each core is assigned a set of seed groups and calculates the best grouping for each of its seed groups.

For example, if the set of seed groups is 4,000 groups and there are 1000 cores, each core will be assigned 4 seed groups. Since calculating groupings later in the series is less computationally intensive (e.g. $group_0$ has to consider 4,000 more groups than $group_{3,999}$), thus each core does not receive consecutive seed groups.

The controller program calculates the groups, weights, and starts the processing cores. When the cores complete calculating the best groupings for their seed groups, the controller program sorts the resulting groupings from highest weight to lowest weight.

In the event that no groupings were found (this can happen if the set of groupings was pruned too much), the algorithm can be repeated with a larger set of groups.

VI. Conclusion and Future Work

Existing heuristic grouping algorithms have not adequately addressed the algorithmic complexity and the topology of the data. The complexity of the problem and the experimental results reported above demonstrate: (1) it is unlikely that a large number of optimal and near-optimal solutions exist, (2) the best solutions are not at the top of well formed hills and thus hill-climbing algorithms are unlikely to find good solutions, (3) it is very unlikely (1 in 10^{12}) existing group formation systems will find good solutions.

The presented grouping algorithm considers the data topology and is thus more likely to find a near-optimal or even an optimal solution. This algorithm has been used successfully to form groups for two software engineering classes. The next step in this research is to demonstrate that the pruning in the presented algorithm effectively finds good groups for several real-life data sets. This will be accomplished by collecting student information from several classes, creating groupings using the described algorithm, and then comparing those grouping to the results from using this algorithm with significantly less pruning (the algorithm

ISBN: 978-988-19252-3-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

will take many weeks to run with less pruning).

REFERENCES

[1] Michelle Craig, Diane Horton, François Pitt, Forming reasonably optimal groups: (FROG), Proceedings of the 16th ACM international conference on Supporting group work, November 07-10, 2010.

[2] Tyson R. Henry. 2013. Creating effective student groups: an introduction to groupformation.org. In Proceeding of the 44th ACM technical symposium on Computer science education (SIGCSE '13). ACM, New York, NY, USA, 645-650. DOI=10.1145/2445196.2445387

[3] Cavanaugh, R., Ellis, M., Layton, R., and Ardis, M. 2004. Automating the process of assigning students to cooperative-learning teams. In Proceedings of the 2004 ASEE Annual Conference. American Society for Engineering Education.

[4] Maria Kyprianidou, Stavros Demetriadis, Andreas Pombortsis, George Karatasios, (2009) "PEGASUS: designing a system for supporting group activity", Multicultural Education & Technology Journal, Vol. 3 Iss: 1, pp.47 – 60

[5] R. A. Layton, M. L. Loughry, M. W. Ohland, and G. D. Rico. Design and validation of a web-based system for assigning members to teams using instructor-specified criteria. In press, Advances in Engineering Education. Advances in Engineering Education, 2(1), 2010, pp. 1–28.

[6] David Meyer. 2009. OptAssign-A web-based tool for assigning students to groups. Comput. Educ. 53, 4 (December 2009), 1104-1119. DOI=10.1016/j.compedu.2009.05.022

[7] Dai-Yi Wang , Sunny S. J. Lin , Chuen-Tsai Sun, DIANA: A computer-supported heterogeneous grouping system for teachers to conduct successful small learning groups, Computers in Human Behavior, v.23 n.4, p.1997-2010, July, 2007. DOI=10.1016/j.chb.2006.02.008

[8] Agustín-Blas, L. E. , Salcedo-Sanz , S., Ortiz-García , E. G., Portilla-Figueras , A., Pérez-Bellido , Á. M., Jiménez-Fernández, S. 2011. Team formation based on group technology: A hybrid grouping genetic algorithm approach, Computers and Operations Research, v.38 n.2, p.484-495.

[9] D. Strnad and N. Guid. 2010. A fuzzy-genetic decision support system for project team formation. Appl. Soft Comput.10, 4 (September 2010), 1178-1187. DOI=10.1016/j.asoc.2009.08.032

[10] Virginia Yannibelli and Analía Amandi. 2012. A deterministic crowding evolutionary algorithm to form learning teams in a collaborative learning context. Expert Syst. Appl. 39, 10 (August 2012), 8584-8592. DOI=10.1016/j.eswa.2012.01.195

[11] S. Graf and R. Bekele. Forming heterogeneous groups for intelligent collaborative learning systems with ant colony optimization. In Proceedings of the 8th International Conference on Intelligent Tutoring Systems (ITS 2006), volume 4053 of Lecture Notes in Computer Science, pages 216--226. Springer, 2006.

[12] Matthew E. Gaston and Marie desJardins. 2005. Agent-organized networks for dynamic team formation. In Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems (AAMAS '05). ACM, New York, NY, USA, 230-237. DOI=10.1145/1082473.1082508

[13] Baykasoglu, A., Dereli, T., Das, S. 2007. Project Team Selection Using Fuzzy Optimization Approach. Cybern. Syst. 38, 2 (February 2007), 155-185. DOI=10.1080/01969720601139041

[14] Feng, B., Jiang, Z., Fan, Z., Fu, N. 2010. A method for member selection of cross-functional teams using the individual and collaborative performances, European Journal of Operational Research, Volume 203, Issue 3, 16 June 2010, Pages 652-661, ISSN 0377-2217, DOI: 10.1016/j.ejor.2009.08.017.

[15] Christodoulopoulos, C.E., Papanikolaou, K. 2007. Investigation of group formation using low complexity algorithms. In: Proc. of PING Workshop.

[16] T. Chen, M. Kwiatkowska, D. Parker, A. Simaitis, Verifying Team Formation Protocols with Probabilistic Model Checking, Proceedings of the 2th International Workshop on Computational Logic in Multi-Agent Systems (CLIMA XII 2011), volume 6814 of LNCS, pages 190-297, Springer. July 2011.

[17] S. Datta, A. Majumder, KVM Naidu, Capacitated Team Formation Problem on Social Networks, Proceedings of the 18th ACM SIGKDD international conference on Knowledge discover and data mining.

[18] Cutshall, R., Gavirneni, S., and Schultz, K. 2007. Indiana University's Kelley School of Business Uses Integer Programming to Form Equitable, Cohesive Student Teams. Interfaces 37, 3 (May 2007), 265-276. DOI=10.1287/inte.1060.0248

[19] P. Brereton and S. Lees, “An Investigation Of Factors Affecting Student Group Project Outcomes,” Proceedings of the 18th Conference on Software Engineering Education and Training, pp. 163-170, 2005.

[20] Lisa Bender, Gursimran Walia, Krishna Kambhampaty, Kendall E. Nygard, and Travis E. Nygard. 2012. Social sensitivity and classroom team projects: an empirical investigation. In Proceedings of the 43rd ACM technical symposium on Computer Science Education (SIGCSE '12). ACM, New York, NY, USA, 403-408.

ISBN: 978-988-19252-3-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)