Intelligence Techniques Are Needed to Further Enhance the Advantage of Groups with Diversity in Problem Solving

Oscar Castillo, Patricia Melin, J. Esteban Gamez, Vladik Kreinovich, and Olga Kosheleva

***Abstract*—In practice, there are many examples when the diversity in a group enhances the group's ability to solve problems – and thus, leads to more efficient groups, firms, schools, etc. Several papers, starting with the pioneering research by Scott E. Page from the University of Michigan at Ann Arbor, provide a theoretical justification for this known empirical phenomenon. However, when the general advise of increasing diversity is transformed into simple-to-follow algorithmic rules (like quotas), the result is not always successful. In this paper, we prove that the problem of designing the most efficient group is computationally difficult (NP-hard). Thus, in general, it is not possible to come up with simple algorithmic rules for designing such groups: to design optimal groups, we need to combine standard optimization techniques with intelligent techniques that use expert knowledge.**

I. Introduction to the problem

In real life, there are many examples that diversity in a group enhances the group's ability to solve problems – and thus, leads to more efficient groups, firms, schools, etc. Several papers, starting with the pioneering research by Scott E. Page from the University of Michigan at Ann Arbor, provide a theoretical justification for this known empirical phenomenon; see, e.g., [2], [6] and references therein. Specifically, these papers have shown that groups of diverse problem solvers *can* outperform groups of high-ability problem solvers.

The word *can* is here (and in the title of the paper [2]) for a good reason: when the general advise of increasing diversity is transformed into simple-to-follow algorithmic rules (like quotas), the result is not always successful.

In this paper, we consider the problem of designing the most efficient group as a precise optimization problem. We show that this optimization problem is computationally difficult (NP-hard). Thus, in general, it is not possible to come up with simple algorithmic rules for designing such groups: to design optimal groups, we need to combine standard optimization techniques with intelligent techniques that use expert knowledge.

Comment. Similar results are known: e.g., the problem of maximizing diversity and the problem of finding a group which is most representative of the population are both NP-hard [1], [4]. In this paper, we extent these results further – from gauging and maintaining the *degree* of diversity to gauging and maintaining the *positive effects* of diversity – such as the increased ability of a group to solve problems.

Oscar Castillo and Patricia Melin are with Tijuana Institute of Technology, mailing address P.O. Box 4207, Chula Vista, CA 91909, USA, emails ocastillo@hafsamx.org and epmelin@hafsamx.org. J. Esteban Gamez, Vladik Kreinovich (Department of Computer Science), and Olga Kosheleva (Department of Teacher Education) are with the University of Texas at El Paso, El Paso, TX 79968, USA, emails vladik@utep.edu and olgak@utep.edu.

This work was supported in part by NSF grant HRD-0734825 and by Grant 1 T36 GM078000-01 from the National Institutes of Health.

II. Towards formulation of the problem in exact terms

Let us assume that we have a population consisting of n individuals. From this population $\{1, \ldots, n\}$, we need to select a group $G \subseteq \{1, \ldots, n\}$ which is the most efficient in solving a given problem.

In mathematical terms, to describe a group G, we must describe, for each individual i ($i = 1, \ldots, n$), whether this individual is selected for this group or not. In computational terms, for each $i = 1, \ldots, n$, we thus need to select a Boolean ("true"-"false") value x_i for which:

- $x_i =$"true" means that we select the i-th individual into the group, and
- $x_i =$"false" means that we do not select the i-th individual into the group.

Inside the computer, "true" is usually represented as 1, while "false" is usually represented as 0. Thus, we can describe each group by selecting, for each $i = 1, \ldots, n$, a value $x_i \in \{0, 1\}$ for which

- $x_i = 1$ means that we select the i-th individual into the group, and
- $x_i = 0$ means that we do not select the i-th individual into the group.

In order to select the most efficient group, we must describe how the group's efficiency p depends on the selections x_i.

For simple mechanical tasks like digging trenches or doing simple menial work, people perform these tasks individually. For such tasks, the efficiency p of a group is simply the sum of the productivity values p_i of all the individuals who form this group G:

$$p = \sum_{i \in G} p_i.$$

In terms of the variables x_i, this formula means that we add p_i if $x_i = 1$ and that we do not add p_i if $x_i = 0$. In other words, this formula can be described as

$$p = \sum_{i=1}^{n} p_i \cdot x_i.$$

In this simple model, the more people work on a project, the larger the productivity.

Most practical problems are not that simple. In solving these problems, interaction between the individuals can enhance their productivity. In mathematical terms, this means that

- in addition to the above terms $p_i \cdot x_i$ which are linear in x_i,
- we also have terms
$$p_{ij} \cdot x_i \cdot x_j, \quad i \neq j,$$
which are quadratic in x_i; these terms describe pair-wise interaction between the individuals;
- we may also have cubic terms
$$p_{ijk} \cdot x_i \cdot x_j \cdot x_k$$
which describe triple interactions,
- and we can also have higher order terms, which describe the effect of larger subgroups.

In other words, in general, the formula describing the productivity of a group takes a more complex form

$$p = \sum_{i=1}^{n} p_i \cdot x_i + \sum_{i \neq j} p_{ij} \cdot x_i \cdot x_j + \ldots.$$

For example, for a group consisting of two individuals, i and j, the productivity is equal to

$$p = p_i + p_j + p_{ij} + \ldots$$

It should be mentioned that interaction is not always helpful. For example, if we are interested in solving a complex problem, and we bring together two individuals with similar ways of thinking and with similar skills, then there is not much that these individuals can learn from each other.

In some cases, they may speed up the process by dividing the testing of possible approaches between themselves. In such cases, they can solve the problem twice faster, the productivity increases twice – so there is, in effect, no interaction terms p_{ij}.

In other cases, when the problem is not easy to subdivide, the fact that we have two similar solvers solving the same problem does not help at all – the overall time is the same as for each individual solver. In this case, $p \approx p_i \approx p_j$ and thus $p < p_i + p_j$, i.e., $p_{ij} < 0$.

On the hand, in a diverse group, individuals complement each other, learn from each other, and as a result, their productivity increases above what would have happened if they worked on their own: $p > p_i + p_j$, so $p_{ij} > 0$.

Such "negative" and "positive" interactions (i.e., $p_{ij} < 0$ and $p_{ij} > 0$) are not just a negative possibility – this is exactly the reason why, as we have mentioned, groups of diverse problem solvers can outperform groups of high-ability problem solvers.

Because of the interaction, the problem of selecting the optimal group becomes non-trivial. In this paper, we show that the problem of selecting an optimal group is computationally difficult (NP-hard). Moreover, we will show that this problem is NP-hard already if we take into account the simplest possible non-linear terms – i.e., quadratic terms. In other words, the problem becomes NP-hard already for the following productivity expression:

$$p = \sum_{i=1}^{n} p_i \cdot x_i + \sum_{i \neq j}^{n} p_{ij} \cdot x_i \cdot x_j.$$

So, we arrive at the following problem.

Definition 1. *By a* problem of selecting the most efficient group*, we mean the following problem. We are given:*

- *an integer $n > 0$;*
- *rational numbers $p_1, \ldots, p_n$, and*
- *rational numbers r_{ij}, $1 \leq i, j \leq n$, $i \neq j$.*

We must find the combination of n values
$x_1 \in \{0, 1\}, \ldots, x_n \in \{0, 1\}$ for which the expression

$$p = \sum_{i=1}^{n} p_i \cdot x_i + \sum_{i \neq j} p_{ij} \cdot x_i \cdot x_j$$

is the largest possible.

Instead of trying to find the most efficient group, we can also formulate a less ambitious problem of finding a group with a given efficiency.

Definition 2. *By a* problem of selecting a group with a given efficiency*, we mean the following problem. We are given:*

- *an integer $n > 0$;*
- *rational numbers $p_1, \ldots, p_n$,*
- *rational numbers r_{ij}, $1 \leq i, j \leq n$, $i \neq j$, and*
- *a rational value p_0.*

We must find the combination of n values
$x_1 \in \{0, 1\}, \ldots, x_n \in \{0, 1\}$ for which

$$p \stackrel{\text{def}}{=} \sum_{i=1}^{n} p_i \cdot x_i + \sum_{i \neq j} p_{ij} \cdot x_i \cdot x_j \geq p_0.$$

III. Main Results

Proposition 1. *The problem of selecting the most efficient group is NP-hard.*

Proposition 2. *The problem of selecting a group with a given efficiency is NP-hard.*

The following sections contain a brief reminder of what NP-hardness means and the proof of the statements.

IV. What is NP-Hardness: a Brief Informal Reminder

Informally, a problem $\mathcal{P}_0$ is called *NP-hard* if it is at least as hard as all other problems from a certain reasonable class. Let us describe this notion in more detail.

A. When is an Algorithm Feasible?

The notion of NP-hardness is related to the known fact that some algorithms are feasible and some are not. Whether an algorithm is feasible or not depends on how many computational steps it needs.

For example, if for some input x of length $\text{len}(x) = n$, an algorithm requires 2^n computational steps, then for an input of a reasonable length $n \approx 300$, we would need 2^{300} computational steps. Even if we use a hypothetical computer for which each step takes the smallest physically possible time (the time during which light passes through the smallest known elementary particle), we would still need more computational steps than can be performed during the (approximately 20 billion years) lifetime of our Universe.

A similar estimate can be obtained for an arbitrary algorithm whose running time $t(n)$ on inputs of length n grows at least as an exponential function, i.e., for which, for some $c > 0$, $t(n) \geq \exp(c \cdot n)$ for all n. As a result, such algorithms (called *exponential-time*) are usually considered *not feasible*.

Comment. The fact that an algorithm is not feasible, does not mean that it can never be applied: it simply means that there are cases when its running time will be too large for this algorithm to be practical; for other inputs, this algorithm can be quite useful.

On the other hand, if the running time grows only as a polynomial of n (i.e., if an algorithm is *polynomial-time*), then the algorithm is usually quite feasible.

As a result of the above two examples, researchers have arrived at the following idea: An algorithm $\mathcal{U}$ is called *feasible* if and only if it is *polynomial-time*, i.e., if and only if there exists a polynomial $P(n)$ such that for every input x of length $\text{len}(x)$, the computational time $t_{\mathcal{U}}(x)$ of the algorithm $\mathcal{U}$ on the input x is bounded by $P(\text{len}(x))$: $t_{\mathcal{U}}(x) \leq P(\text{len}(x))$.

In most practical cases, this idea *adequately* describes our intuitive notion of feasibility: *polynomial-time* algorithms are usually *feasible*, and *non-polynomial-time* algorithms are usually *not feasible*. However, the reader should be warned that in some (rare) cases, it does not work:

- Some algorithms are polynomial-time but not feasible: e.g., if the running time of an algorithm is $10^{300} \cdot n$, this algorithm is polynomial-time, but, clearly, not feasible.
- Vice versa, there exist algorithms whose computation time grows, say, as $\exp(0.000\ldots01 \cdot \text{len}(x))$. Legally speaking, such algorithms are exponential time and thus, not feasible, but for all practical purposes, they are quite feasible.

It is therefore desirable to look for a *better* formalization of feasibility, but as of now, "polynomial-time" is the best known description of feasibility.

Definition 3. *An algorithm U is called* feasible *if there exists a polynomial $P(n)$ such that for every input x, the running time $t_U(x)$ of this algorithm does not exceed $P(\text{len}(x))$, where by $\text{len}(x)$, we denoted the length of the input x (i.e., the number of bits that form this input).*

B. When is a Problem Tractable?

At first glance, now, that we have a definition of a feasible algorithm, we can describe which problems are tractable and which problems are intractable: If there exists a polynomial-time algorithm that solves all instances of a problem, this problem is tractable, otherwise, it is intractable.

In some cases, this ideal solution is possible, and we either have an explicit polynomial-time algorithm, or we have a proof that no polynomial-time algorithm is possible.

Unfortunately, in many cases, we do not know whether a polynomial-time algorithm exists or not. This does not mean, however, that the situation is hopeless: instead of the missing *ideal* information about intractability, we have another information that is almost as good.

Namely, for some cases, we do not know whether the problem can be solved in polynomial time or not, but we do know that this problem is as hard as practical problems can get: if we can solve *this* problem easily, then we would have an algorithm that solves *all* problems easily, and the existence of such universal solves-everything-fast algorithm is very doubtful. We can, therefore, call such "hard" problems *intractable*.

In order to formulate this notion in precise terms, we must describe what we mean by a problem, and what we mean by the ability to *reduce* other problems to this one.

What is a practical problem? When we say that there is a practical problem, we usually mean that:

- we have some information (we will denote its computer representation by x), and
- we know the relationship $R(x, y)$ between the known information x and the desired object y.

In the computer, everything is represented by a binary sequence (i.e., sequence of 0's and 1's), so we will assume that x and y are binary sequences.

In this section, we will trace all the ideas on two examples, one taken from mathematics and one taken from physics.

- (Example from *mathematics*) We are given a mathematical statement x. The desired object y is either a proof of x, or a "disproof" of x (i.e., a proof of "not x"). Here, $R(x, y)$ means that y is a proof either of x, or of "not x".
- (Example from *physics*) x is the results of the experiments, and the desired y is the formula that fits all these data. Imagine that we have a series of measurements of voltage and current: e.g., x consists of the following pairs $(x_1^{(k)}, x_2^{(k)})$, $1 \leq k \leq 10$: $(1.0, 2.0), (2.0, 4.0), \ldots, (10.0, 20.0)$; we want to find a formula that is consistent with these experiments (e.g., y is the formula $x_2 = 2 \cdot x_1$).

For a problem to be practically meaningful, we must have a way to check whether the proposed solution is correct. In other words, we must assume that there exists a feasible algorithm that checks $R(x, y)$ (given x and y). If no such feasible algorithm exists, then there is no criterion to decide whether we achieved a solution or not.

Another requirement for a real-life problem is that in such problems, we usually know an *upper bound* for the length $\mathrm{len}(y)$ of the description of y. In the above examples:

- In the *mathematical* problem, a proof must be not too huge, else it is impossible to check whether it is a proof or not.
- In the *physical* problem, it makes no sense to have a formula $x_2 = f(x_1, C_1, \ldots, C_{40})$ with, say, 40 parameters to describe the results $(x_1^{(1)}, x_2^{(1)}), \ldots, (x_1^{(10)}, x_2^{(10)})$ of 10 experiments, for two reasons:
  - First, one of the goals of physics is to discover the laws of nature. If the number of parameters exceeds the number of experimental data, then no matter what dependency $f(x_1, C_1, \ldots)$ we choose, in order to determine C_i, we have, say, 10 equations with 40 unknowns. Such under-determined system usually has a solution, so the fact that, say, a linear formula with many parameters fits all the experimental data does not mean that the dependency is proven to be linear: a quadratic or cubic formula with as many parameters will fit the same data as well.
  - Second, another goal of physics (definitely related to the first one) is to find a way to *compress* the data, so that we will not need to store all billions of experimental results in order to make predictions. A dependency y that requires more storage space than the original data x is clearly not satisfying this goal.

In all cases, it is necessary for a user to be able to read the desired solution symbol-after-symbol, and the time required for that reading must be feasible. In the previous section, we have formalized "feasible time" as a time that is bounded by some polynomial of $\mathrm{len}(x)$. The reading time is proportional to the length $\mathrm{len}(y)$ of the answer y. Therefore, the fact the reading time is bounded by a polynomial of $\mathrm{len}(x)$ means that the length of the output y is also bounded by some polynomial of $\mathrm{len}(x)$, i.e., that $\mathrm{len}(y) \le P_L(\mathrm{len}(x))$ for some polynomial P_L.

So, we arrive at the following formulation of a problem:

Definition 4. *By a* general practical problem *(or simply a* problem*, for short), we mean a pair $\langle R, P_L\rangle$, where $R(x, y)$ is a feasible algorithm that transforms two binary sequences into a Boolean value ("true" or "false"), and P_L is a polynomial.*

Definition 5. *By an* instance *of a (general) problem $\langle R, P_L\rangle$, we mean the following problem:*

GIVEN: a binary sequence x.
GENERATE

- *either y such that $R(x, y)$ is true and $\mathrm{len}(y) \le P_L(\mathrm{len}(x))$,*
- *or, if such a y does not exist, a message saying that there are no solutions.*

For example, for the general mathematical problem described above, an instance would be: given a statement, find its proof or disproof.

Comments. What we called "general practical problems" is usually described as "problems from the class NP" (to separate them from more complicated problems in which the solution may not be easily verifiable). Problems for which there is a feasible algorithm that solves all instances are called *tractable*, *easily solvable*, or "problems from the class P" (P from *P*olynomial). It is widely believed that not all (general practical) problems are easily solvable (i.e., that NP≠P), but it has never been proved.

One way to solve an NP problem is to check $R(x, y)$ for all binary sequences y with $\mathrm{len}(y) \le P_L(\mathrm{len}(x))$. This algorithm (called *British Museum* algorithm) requires $2^{P_L(\mathrm{len}(x))}$ checks. This algorithm takes exponential time and is therefore, not feasible.

C. Reducing a Problem to Another One

Let us start with an example. Suppose that we can have an algorithm that checks whether a given system of linear inequalities

$$a_{i1} \cdot x_1 + \ldots + a_{im} \cdot x_m \ge b_i, \quad 1 \le i \le n,$$

with known a_{ij} and b_i, has a solution. A problem of checking whether a given system of inequalities *and equalities* $c_{k1} \cdot x_1 + \ldots + c_{km} \cdot x_m = d_k$ is consistent can be *reduced* to the problem of checking inequalities if we replace each equality by two inequalities: $c_{k1} \cdot x_1 + \ldots + c_{km} \cdot x_m \ge d_k$ and $(-c_{k1}) \cdot x_1 + \ldots + (-c_{km}) \cdot x_m \ge -d_j$ (the latter being equivalent to $c_{k1} \cdot x_1 + \ldots + c_{km} \cdot x_m \le d_k$).

In general, we can say that a problem $\mathcal{P} = \langle R, P_L\rangle$ can be *reduced* to a problem $\mathcal{P}' = \langle R', P'_L\rangle$ if there exist three feasible algorithms U_1, U_2, and U_3 with the following properties:

- The (feasible) algorithm U_1 transforms each input x of the first problem into an input of the second problem.
- The (feasible) algorithm U_2 transforms each solution y of the first problem into the solution of the corresponding case of the second problem: i.e., if $R(x, y)$ is true, then $R'(U_1(x), U_2(y))$ is also true.
- The (feasible) algorithm U_3 transforms each solution y' of the corresponding instance of the second problem into the solution of the first problem: i.e., if $R'(U_1(x), y')$ is true, then $R(x, U_3(y'))$ is also true.

(In the above example, U_1 transforms each equality into two inequalities, and U_2 and U_3 simply do not change the values x_i at all.)

If there exists a reduction, then an instance x of the first problem is solvable if and only if the corresponding instance $U_1(x)$ of the second problem is solvable. Moreover, if we can actually solve the second instance (and find a solution y'), we will then be able to find a solution to the original instance x of the first problem (as $U_3(y')$). Thus, if we have a *feasible* algorithm for solving the second problem, we would thus design a *feasible* algorithm for solving the first problem as well.

Comment. We only described the simplest way of reducing one problem to another one: when a *single* instance of the first problem is reduced to a *single* instance of the second problem. In some cases, we cannot reduce to a *single* case, but we can reduce to *several* cases, solving which helps us solve the original instance of the first problem.

Definition 6.

- *A problem (not necessarily from the class NP) is called* NP-hard *if every problem from the class NP can be reduced to it.*
- *If a problem from the class NP is NP-hard, it is called* NP-complete*.*

If a problem $\mathcal{P}$ is NP-hard, then every feasible algorithm for solving *this* problem $\mathcal{P}$ would lead to feasible algorithms for solving *all* problems from the class NP, and this is generally believed to be hardly possible.

- For example, mathematicians believe that not only there is *no algorithm* for checking whether a given statement is provable or not (the famous Gödel's theorem has proven that), but also they believe that there is *no feasible way* to find a proof of a given statement even if we restrict the lengths of possible proofs. (In other words, mathematicians believe that computers cannot completely replace them.)
- Similarly, physicists believe that what they are doing cannot be completely replaced by computers.

In view of this belief, NP-hard problems are also called *intractable*.

Comment. It should be noted that although most scientists *believe* that intractable problems are not feasible, we still *cannot prove* (or disprove) this fact. If a NP-hard problem *can* be solved by a feasible algorithm, then (by definition of NP-hardness) *all* problems from the class NP will be solvable by feasible algorithms and thus, P=NP. Vice versa, if P=NP, then all problems from the class NP (including all NP-complete problems) can be solved by polynomial-time (feasible) algorithms.

So, if P≠NP (which is a common belief), then the fact that the problem is NP-hard means that *no matter what algorithm we use, there will always be some cases for which the running time grows faster than any polynomial*. Therefore, for these cases, the problem is truly intractable.

D. Examples of NP-Hard Problems

Historically the NP-complete problem proved to be NP-complete was the so-called *propositional satisfiability* (*3-SAT*) problem for $3-$CNF formulas.

This problem consists of the following: Suppose that an integer v is fixed, and a formula F of the type $F_1 \& F_2 \& \ldots \& F_k$ is given, where each of the expressions F_j has the form $a \vee b$ or $a \vee b \vee c$, and a, b, c are either the variables $z_1, \ldots, z_v$, or their negations $\neg z_1, \ldots, \neg z_v$ (these $a, b, c, \ldots$ are called *literals*)

For *example*, we can take a formula $(z_1 \vee \neg z_2) \& (\neg z_1 \vee z_2 \vee \neg z_3)$.

If we assign arbitrary Boolean values ("true" or "false") to v variables $z_1, \ldots, z_v$, then, applying the standard logical rules, we get the truth value of F. We say that a formula F is *satisfiable* if there exist truth values $z_1, \ldots, z_v$ for which the truth value of the expression F is "true". The problem is: given F, check whether it is satisfiable.

In the *subset sum* problem, given n integers $s_1, \ldots, s_n$, we must check whether there exist values $x_1, \ldots, x_n \in \{-1, 1\}$ for which $s_1 \cdot x_1 + \ldots + s_n \cdot x_n = 0$.

V. Towards a Proof: How NP-Hardness Is Usually Proved

The original proof of NP-hardness of certain problems $\mathcal{P}_0$ is rather complex, because it is based on explicitly proving that *every* problem from the class NP can be reduced to the problem $\mathcal{P}_0$. However, once we have proven NP-hardness of a problem $\mathcal{P}_0$, the proof of NP-hardness of other problems $\mathcal{P}_1$ is much easier.

Indeed, from the above description of a reduction, one can easily see that reduction is a transitive relation: if a problem $\mathcal{P}$ can be reduced to a problem $\mathcal{P}_0$, and the problem $\mathcal{P}_0$ can be reduced to a problem $\mathcal{P}_1$, then, by combining these two reductions, we can prove that $\mathcal{P}$ can be reduced to $\mathcal{P}_1$.

Thus, to prove that a new problem $\mathcal{P}_1$ is NP-hard, it is sufficient to prove that one of the known NP-hard problems $\mathcal{P}_0$ can be reduced to this problem $\mathcal{P}_1$. Indeed, since $\mathcal{P}_0$ is NP-hard, every other problem $\mathcal{P}$ from the class NP can be reduced to this problem $\mathcal{P}_0$. Since $\mathcal{P}_0$ can be reduced to $\mathcal{P}_1$, we can now conclude, by transitivity, that every problem $\mathcal{P}$ from the class NP can be reduced to this problem $\mathcal{P}_1$ – i.e., that the problem $\mathcal{P}_1$ is indeed NP-hard.

Comment. As a consequence of the definition of NP-hardness, we can conclude that if a problem $\mathcal{P}_0$ is NP-hard, then every more general problem $\mathcal{P}_1$ is also NP-hard.

Indeed, the fact that $\mathcal{P}_0$ is NP-hard means that every instance p of every problem $\mathcal{P}$ can be reduced to some instance p_0 of the problem $\mathcal{P}_0$. Since the problem $\mathcal{P}_1$ is more general than the problem $\mathcal{P}_0$, every instance p_0 of the problem $\mathcal{P}_0$ is also an instance of the more general problem $\mathcal{P}_1$.

Thus, every instance p of every problem $\mathcal{P}$ can be reduced to some instance p_0 of the problem $\mathcal{P}_1$ – i.e., that the more general problem $\mathcal{P}_1$ is indeed NP-hard.

VI. Reduction in Our Proof: to Subset Sum, a Known NP-Hard Problem

We prove NP-hardness of our problem by reducing a known NP-hard problem to it: namely, a *subset sum* problem, in which we are given n positive integers $s_1, \ldots, s_n$, and we must find the signs $\varepsilon_i \in \{-1, 1\}$ for which

$$\sum_{i=1}^{n} \varepsilon_i \cdot s_i = 0;$$

see, e.g., [7].

A reduction means that to every instance $s_1, \ldots, s_n$ of the subset sum problem, we must assign (in a feasible, i.e., polynomial-time way) an instance of our problem in such a way that the solution to the new instance will lead to the solution of the original instance.

VII. Reduction: Idea

In our reduction, we would like to transform each variable ε_i from the subset sum problem into a variable x_i from our problem, so that our problem (formulated in terms of x_i) is optimal if and only if the original problem has a solution.

For that, we need to transform each variable x_i which takes the values 0 and 1 into a variable ε_i that takes values -1 and 1 (and vice versa). The simplest way to perform this reduction is to take a linear function $\varepsilon_i = a \cdot x_i + b$, where the coefficients a and b are selected in such as way that $a \cdot 0 + b = -1$ and $a \cdot 1 + b = 1$. In other words, we have $b = -1$ and $a + b = 1$. Substituting $b = -1$ into the equation $a + b = 1$, we conclude that $a = 2$, i.e., that

$$\varepsilon_i = 2 \cdot x_i - 1.$$

Let us select an integer $p_0 > 0$ and consider the formula

$$p = p_0 - \left(\sum_{i=1}^n \varepsilon_i \cdot s_i\right)^2.$$

This expression is always $\le p_0$, and it attains the value p_0 if and only if $\sum\limits_{i=1}^n \varepsilon_i \cdot s_i = 0$. In terms of x_i, we have

$$p = p_0 - \left(\sum_{i=1}^n (2 \cdot x_i - 1) \cdot s_i\right)^2,$$

i.e.,

$$p = p_0 - \left(\sum_{i=1}^n x_i \cdot (2 \cdot s_i) - s_0\right)^2,$$

where we denoted

$$s_0 \overset{\text{def}}{=} \sum_{i=1}^n s_i.$$

By using the formula for the square of the difference, we conclude that

$$p = p_0 - \left(\sum_{i=1}^n x_i \cdot (2 \cdot s_i)\right)^2 +$$

$$2 \cdot s_0 \cdot \sum_{i=1}^n x_i \cdot (2 \cdot s_i) - s_0^2,$$

i.e.,

$$p = p_0 - \left(\sum_{i=1}^n x_i \cdot (2 \cdot s_i)\right)^2 +$$

$$\sum_{i=1}^n x_i \cdot (4 \cdot s_0 \cdot s_i) - s_0^2.$$

The square of the sum takes the form

$$\left(\sum_{i=1}^n x_i \cdot (2 \cdot s_i)\right)^2 =$$

$$\sum_{i=1}^n x_i^2 \cdot (4 \cdot s_i^2) + \sum_{i \ne j} (4 \cdot s_i \cdot s_j) \cdot x_i \cdot x_j.$$

Since $x_i = 0$ or $x_i = 1$, we always have $x_i^2 = x_i$ and thus,

$$\left(\sum_{i=1}^n x_i \cdot (2 \cdot s_i)\right)^2 =$$

$$\sum_{i=1}^n x_i \cdot (4 \cdot s_i^2) + \sum_{i \ne j} (4 \cdot s_i \cdot s_j) \cdot x_i \cdot x_j.$$

Substituting this expression into the above formula for p, we get

$$p = p_0 - \sum_{i=1}^n x_i \cdot (4 \cdot s_i^2) - \sum_{i \ne j} (4 \cdot s_i \cdot s_j) \cdot x_i \cdot x_j +$$

$$\sum_{i=1}^n x_i \cdot (4 \cdot s_0 \cdot s_i) - s_0^2.$$

By grouping together terms independent on x_i and terms proportional to p_i, we get

$$p = (p_0 - s_0^2) + \sum_{i=1}^n x_i \cdot (4 \cdot s_0 \cdot s_i - 4 \cdot s_i^2) +$$

$$\sum_{i \ne j} (-4 \cdot s_i \cdot s_j) \cdot x_i \cdot x_j.$$

Thus, if we choose

$$p_0 = s_0^2,$$

then the above expression takes the desired form

$$p = \sum_{i=1}^n p_i \cdot x_i + \sum_{i \ne j} p_{ij} \cdot x_i \cdot x_j,$$

with

$$p_i = 4 \cdot s_0 \cdot s_i - 4 \cdot s_i^2$$

and

$$p_{ij} = -4 \cdot s_i \cdot s_j.$$

VIII. Resulting Reduction

To each particular case of the subset sum problem, described by parameters $s_1, \ldots, s_n$, we assign the following particular case of our problem. First, we compute

$$p_0 = s_0 = \sum_{i=1}^n s_i;$$

then, we compute

$$p_i = 4 \cdot s_0 \cdot s_i - 4 \cdot s_i^2; \quad p_{ij} = -4 \cdot s_i \cdot s_j.$$

For these values, the quadratic function

$$p = \sum_{i=1}^{n} p_i \cdot x_i + \sum_{i \neq j} p_{ij} \cdot x_i \cdot x_j$$

has the form

$$p = p_0 - \left(\sum_{i=1}^{n} \varepsilon_i \cdot s_i \right)^2 ,$$

where $\varepsilon_i = 2 \cdot x_i - 1 \in \{-1, 1\}$.

The above argument shows that for this selection, the quadratic function p attains the value $p_0 = s_0$ if and only if the original instance of the subset sum problem $\sum\limits_{i=1}^{n} \varepsilon_i \cdot s_i = 0$ has a solution with $\varepsilon_i \in \{-1, 1\}$, and thus, with

$$x_i = \frac{\varepsilon_i + 1}{2} \in \{0, 1\}.$$

The reduction is proven, so our problem is indeed NP-hard.

Comment. Strictly speaking, we have proved NP-hardness of a specific choice of the quadratic function $p(x_1, \ldots, x_n)$. However, we have already mentioned earlier that if a problem $\mathcal{P}_0$ is NP-hard, then a more general problem $\mathcal{P}_1$ is NP-hard as well. Thus, we have indeed proved that the (more general) problem is also NP-hard.

IX. Conclusions

One of the applications of fuzzy techniques is to formalize the meaning of words from natural language such as "efficient", "diverse", etc. The main idea behind fuzzy techniques is that they formalize expert knowledge expressed by words from natural language; see, e.g, [3], [5].

In this paper, we have shown that if we do not use this knowledge, i.e., if we only use the data, then selecting the most efficient group (or even selecting a group with a given efficiency) becomes a computationally difficult (NP-hard) problem. Thus, the need to select such groups in reasonable time justifies the use of fuzzy (intelligent) techniques – and, moreover, the need to combine intelligent techniques with more traditional optimization techniques.

References

[1] J. E. Gamez, F. Modave, and O. Kosheleva, "Selecting the Most Representative Sample is NP-Hard: Need for Expert (Fuzzy) Knowledge", *Proceedings of the IEEE World Congress on Computational Intelligence WCCI'2008*, Hong Kong, China, June 1–6, 2008, pp. 1069–1074.

[2] L. Hong and S. E. Page, "Groups of diverse problem solvers can outperform groups of high-ability problem solvers", *Proceedings of the National Academy of Sciences*, 2004, Vol. 101, No. 46. pp. 16385–16389.

[3] G. Klir and B. Yuan, *Fuzzy sets and fuzzy logic: theory and applications.* Prentice Hall, Upper Saddle River, New Jersey, 1995.

[4] C. C. Kuo, F. Glover, and K. S. Dhir, "Analyzing and modeling the maximum diversity problem by zero-one programming", *Decision Sciences*, vol. 24, no. 6, pp. 1171–1185, 1993.

[5] H. T. Nguyen and E. A. Walker, *A first course in fuzzy logic*, CRC Press, Boca Raton, Florida, 2005.

[6] S. E. Page, *The Difference: How the Power of Diversity Creates Better Groups, Firms, Schools, and Societies*, Princeton University Press, Princeton, New Jersey, 2007.

[7] C. H. Papadimitriou, *Computational Complexity*, Addison Wesley, San Diego, 1994.