Journal of Advances and Scholarly Researches in Allied Education
Vol. 16, Issue No. 4, March-2019, ISSN 2230-7540

A Study of Operations Research Solutions Techniques in Combinatorial Problem towards Constraint Programming

Mr. Akash Pandey[1]* Dr. Umesh Kumar Gupta[2]

[1] Research Scholar, Himalayan University Itanagar, Arunachal Pradesh

[2] Associate Professor, M. G. P. G. College Gorakhpur (U.P.)

Abstract – Many numerical identities are proved applying clever but informal combinatorial arguments. A formal representation is presented to prove these identities closely following these arguments. The main formal tool used to that end is, operationally, an abstract and axiomatic generalization of the Sigma notation (Σ), which is utilized for expressing and manipulating summations and counts. Operational properties are provided with algebraic properties that make it possible to perform different naturally important operations. In this paper we present a formal version and some more examples that show how to practically interpret combinatorial arguments used in literature. We present typical combinatorial evidence of the inclusion-exclusion theorem. Combinatory evidence of numerical identity is either that both sides of the given equation count the very same kinds of objects in two different ways or show a bijection between the sets that show up on each side of the equation. The two expressions must therefore be the same. The dream most combinatorialists make when they prove their identity is this kind of argument. Such arguments are generally informal and, it is likely to be safe to say, ad hoc, without a unified formal basis.

Key Words – Operations Research, Solutions Techniques, Combinatorial Problem, Constraint Programming

----------X----------

INTRODUCTION

In this article we show a formal display that closely follows the appropriate combinatorial arguments, which proves this identity. After presenting formal notes and their fundamental algebraic transformations, we show how, in practice, combination arguments used in the literature to demonstrate a selection of identities with binomial coefficients and permutations can be formally interpreted. We shall give the typical combination proof of the theorem Inclusion Exclusion in particular in a formal version. Furthermore, some of these calculations illustrate how a recursive definition can be reached for resolving a counter problem, i.e. formally derived a recursive algorithm to solve the problem. The official tool used is known as an operational instrument. In the main, the Sigma (Σ) notation, used to express and manipulate summations and numbers, is an abstract and axiomatic generalization. Operations enable various natural operations with combinatorial significance because of their algebraic properties. Among these algebraic properties are: a way of splitting summaries with different divided separation criteria (a measure called conditioning) and the ability to map or transform the counted elements in a bijection to a variety of elements (basically, count and summation indices) (the essence of a combinatorial argument). In terms of graphical, symbolic and material object count, combinational arguments are generally expressed, such as paths along the rims of a grid, 'montane ranges," matched parenthetic parentheses, grid tiling by blocks, beaded necklaces etc. We modell these objects with discreet mathematical structures like the finite numbers, strings and finite sequences of numbers in order to officially formalize these concepts (possibly considered as circular). This formal device not only offers official support for expressing and reasoning ideas and evidence, but also enables the deduction of recursive functions to identify solutions to count problems. This method can be widespread to produce formal derivations of recursive algorithmic solutions through gradual refining of certain specifications. We also find this approach feasible to make automated evidence control of combinatorial evidence easier.

Schemes for Incorporating OR into CP

CP's unique concept of a constraint governs how OR methods may be imported into CP. In order to reduce variable domains, the most obvious function of an OR method is to apply it to a constraint or subset of limits. If the restrictions include certain linear inequalities, a variable subject to such inequalities can be minimized or maximized, thus possibly reducing the domain of the variable. The problem of minimizing or maximizing is a linear programming (LP) issue, an OR staple. This is one example of OR's most common scheme: to relax the CP problem, in the form of an OR model, like an LP model. Relaxation solution helps to reduce the domain or guide the search. Other OR models that can play this role include linear (MILP) models (which can be relaxed by themselves), relaxation from Lagrangean and dynamic programming models. OR has also developed expert relieves for a wide range of ordinary situations and offers tools to relax globally.

A CP solver has several advantages of relaxation. (a) it may tighten a variable boundary. (b) In the original problem, its solution may become feasible. (c) If not, the solution may be a promising guide for the search. (d) One solution can allow domains to be filtered in other ways, for example through cost reduction or Lagrange multipliers, or by dynamical programming examinations of state space. (e) The solution can deliver a binding value for optimal pruning of the search tree for optimisation problems. (e) More generally, by combining several limitations relaxed in a single OR-based relaxation process, global problems that are only partly captured by constraint propagation can be taken advantage of. The problem is broken down by other hybridisation schemes, so the CP and OR can attack the parts of the problem they most suit. To date, branch and price algorithms and generalizations of Benders degradation have been the most important schemes. Branch and price based on CP generate a column, i.e., the variables that are to be dynamically added to enhance the solution during a branch search. For the "row generation," decomposition is often done using CP; in other words, restrictions (goods) are generated that guide the main search method.

Table 1: Sampling of computational results for methods that combine CP and OR

Problem	Contribution to CP	Speedup
CP plus relaxations similar to those used in MILP		
Lesson timetabling [51]	Reduced-cost variable fixing using an assignment problem relaxation.	2 to 50 times faster than CP.
Minimizing piecewise linear costs [105]	Convex hull relaxation of piecewise linear function.	2 to 200 times faster than MILP. Solved two instances that MILP could not solve.
Boat party & flow shop scheduling [77]	Convex hull relaxation of disjunctions, covering inequalities.	Solved 10-boat instance in 5 min that MILP could not solve in 12 hours. Solved flow shop instances 3 to 4 times faster.
Product configuration [121]	Convex hull relaxation of element constraints, reduced cost variable fixing.	30 to 40 times faster than MILP (which was faster than CP).
Automatic digital recording [113]	Lagrangean relaxation.	1 to 10 times faster than MILP (which was faster than CP).
Stable set problems [66]	Semi-definite programming relaxation.	Significantly better suboptimal solutions than CP in fraction of the time.
Structural design [23]	Linear quasi-relaxation of nonlinear model with discrete variables.	Up to 600 times faster than MILP. Solved 2 problems in < 6 min that MILP could not solve in 20 hours.
Scheduling with earliness and tardiness costs [14]	LP relaxation.	Solved 67 of 90 instances, while CP solved only 12.
CP-based branch and price		
Traveling tournament scheduling [44]	Branch-and-price framework.	First to solve 8-team instance.
Urban transit crew management [133]	Branch-and-price framework.	Solved problems with 210 trips, while traditional branch and price could accommodate only 120 trips.
Benders-based integration of CP and MILP		
Min-cost multiple machine scheduling [81]	MILP master problem, CP feasibility subproblem.	20 to 1000 times faster than CP, MILP.
Min-cost multiple machine scheduling [120]	Updating of single MILP master (branch and check).	Additional factor of 10 over [81].
Polypropylene batch scheduling [122]	MILP master problem, CP feasibility subproblem.	Solved previously insoluble problem in 10 min.
Call center scheduling [16]	CP master, LP subproblem.	Solved twice as many instances as traditional Benders.
Min cost and min makespan planning & cumulative sched. [71]	MILP master problem, CP optimization subproblem.	100 to 1000 times faster than CP, MILP. Solved significantly larger instances.
Min no. late jobs and min tardiness planning & cumulative sched. [72]	MILP master problem, CP optimization subproblem with LP relaxation.	Min late jobs 100-1000 times faster than MILP, CP; min tardiness significantly faster, better solutions when suboptimal.

PRELIMINARIES

Throughout this text, a one-argument function application over a variable or a constant is denoted by an infix dot '.'. To denote multiplications, generally, we will avoid the usual juxtaposition convention by using an explicit infix dot sign (·) due to the confusion caused by naming variables with more than one letter. By true and f alse , we denote, besides the boolean values, the obvious 0-ary constant predicates. The following logical connectives are listed in order of decreasing binding power (those listed as a pair has the same precedence): ¬ denotes negation, ∨ and ∧ denote disjunction and conjunction respectively, and ≡ and 6≡ denote equivalence and discrepancy respectively. As usual, the symbols ∀ and ∃ denote the universal and the existential quantifier respectively.

NOTATION	MEANING
$set\ A$	Type defining the subsets of (finite) set A.
$\|A\|$	Cardinal of (finite) set A.
$[a:b]$	$\{z \in \mathbb{Z} \mid a \le z \le b\}$ for a, b integers.
$[n]$	abbreviation of $[1:n]$ for n positive integer.
$[a:b)$	$\{z \in \mathbb{Z} \mid a \le z < b\}$ for a, b integers.
$s \in set[a:b]$	Declares that s is a subset of $[a:b]$.
$s \subseteq [a:b]$	Abbreviation of $s \in set[a:b]$.
$sq\ A$	Finite sequences containing elements in set A.
$bit(n)$	Sequences of length n with elements in $\{0, 1\}$.
$pm[n]$	Non-repeating sequences of all elements in $[1:n]$.
$[\,]$	The empty sequence
$\|s\|$	Length of sequence s.
$\langle a \rangle$	sequence consisting of just element a.
$s.k$	Element of sequence s in position k. (A sequence may be thought as function defined on $[1:\|s\|]$)
$s \frown t$	Concatenation of sequences s and t.
$s[a:b]$	Segment of s covering positions in $[a:b]$. If $a > b$, $s[a:b] = [\,]$.
$s[a:b)$	Segment of s covering positions in $[a:b)$.

For set operations, we use the usual symbols for union (∪), intersection (∩), membership (∈) and inclusion (⊆). All other notations will be defined in the place where they are used.

A. Operationals

Summations may be interpreted as iterations of the addition operation. This view of iterated sums can be generalized to any associative and commutative operator ⊕ with advantages of unified notation and calculation through their generalized axiomatic description. The expression

$$(\oplus i \in D : R.i : P.i).$$

denotes the iteration of ⊕ over the values of the expression P.i for all the values of i belonging to the domain (or type) D and satisfying the range condition R.i. We call this kind of expression an operational on ⊕. These expressions are also known as quantifications. Observe that in order to avoid an accumulation of parenthesis, we sometimes write (when it does not lead to confusion) R.i to denote application of expression R (considered as a function) to argument i.

Conventional Notation	As an operational
$\Sigma_{i=1}^{n} i^2$	$(+i : 1 \le i \le n : i^2)$
$\Pi_{i=1}^{n} (x+i)$	$(\cdot i : 1 \le i \le n : x+i)$
$max_{i=1}^{n} i^2$	$(\uparrow i : 1 \le i \le n : i^2)$
$\bigcup_{i=1}^{n} S_i$	$(\bigcup i : 1 \le i \le n : S.i)$
$\bigcap_{i=1}^{n} S_i$	$(\bigcap i : 1 \le i \le n : S.i)$

In order to keep some resemblance with the traditional notation we allow to write $(\Sigma i \in D : R.i : P.i)$ instead of $(+i \in D : R.i : P.i)$ and $(\Pi i \in D : R.i : P.i)$ instead of $(\cdot i \in D : R.i : P.i)$. To deal with a false range R.i, it becomes necessary that operator ⊕ has an identity in addition to the symmetry and associativity properties. The table above illustrates the unifying effect of this notation.

Any occurrence of index i in the expressions P.i and R.i above, are said to be bound by the operational. Strictly, index i should be always annotated with a type to indicate the range of values it may assume; however, to avoid cumbersome repetitions, we may state just once the type of a certain index in the context of a calculation. In addition, the range is sometimes omitted when it is refers to the entire domain of variation of i; formally, this correspond to have the predicate true as a range. In this case the form of the operational is

$$(\oplus i \in D :: P.i).$$

One more thing, the counting operational is somewhat different from the others. We introduce it as follows.

$$(\#i \in D : R.i : F.i)$$

could be considered 'syntactic sugar' for

$$(\Sigma i \in D : R.i \wedge F.i : 1)$$

Similarly, (#i ∈ D :: R.i) is defined as (Σi ∈ D : R.i : 1) or |{i ∈ D | R.i}| (the cardinal of the set of elements in D that fulfill property R).

B. Some algebraic properties of operationals.

The operational unifying effect is not limited to notation. It is possible to give operations an abstract treatment in accordance with calculation style in order to easily manipulate the algebraic. Due to the fact that operations are defined as iterating applications of an associative and switched binary operator, properties are generalized which are generally only summaries. These properties are achieved every time the operations are well defined, even if an endless number of operations are involved. For example, if they converge in the case of summations.

Being ⊕ an arbitrary associative commutative binary operator defined and taking values on a domain D, the following properties are valid for operationals of the form (⊕i : R : T) (with R, S Boolean expressions, and T an expression taking values on D). In the following, we suppose as understood that indices i, j take all their values on D.

Splitting.
$(\oplus i : R \vee S : T) = (\oplus i : R : T) \oplus (\oplus i : S : T)$
whenever ranges R and S are disjoint. Formally, $R \neq S$.

Mapping.
$(\oplus i \in D : R : T) = (\oplus j \in E : R[i := f.j] : T[i := f.j])$
whenever f is a bijective function from E to D such

that $i=f.j$ (and then $j=f^{-1}(i)$) for all i in D fulfilling R.

Nesting. $(\oplus i,j : R \wedge S : T) = (\oplus i : R : (\oplus j : S : T))$. This requires index j not appearing in R as a free (unbound) variable.

Empty Range. $(\oplus i : false : T) = u.$ This rule requires $\oplus$ to have an identity element u.

One Point. $(\oplus i : i=K : T) = T[i:=K]$, where $T[i:=K]$ is the expression obtained from substituting K for all occurrences of i in T.

For the counting operational fulfills similar properties, here R, S, T must be Boolean expressions.

Splitting. $(\#i : R \vee S : T) = (\#i : R : T) + (\#i : S : T)$ whenever ranges R and S are disjoint.

FORMAL VERSIONS OF SOME COMBINATORIAL PROOFS

In this section we present different examples of calculational proofs of numerical identities inspired by the corresponding combinatorial proofs as they appear in the literature. For the sake of easing human reading, these proofs are rigorous but not entirely formal. For instance, the mention of the axioms for operationals in the explanations for each step of our calculations just hints the main ideas to justify them. In particular, we leave to the reader to justify that in the mapping steps, the corresponding transformations are in fact bijections from one range of values to the other.

A. Counting Permutations

We define Pn as the number of permutations of the elements in [n]. Formally, we represent a permutation in $[n].$ as a non-repeating sequence q in sq[n] of length n, that encodes a bijection mapping any $i \in [n]$ to $q.i.$. Consequently, we define pm[n] as $\{s \in sq[n] \mid nrep.s \wedge |s|=n\}$, where nrep.s means that the sequence s does not repeat its elements.

For the sake of easy manipulation of permutations, we define, for a (finite) sequence s of integers, its normalization with respect to the deletion of its element in position $(i : 1 \le i \le |s|)$, as follows

$$nrm(s,i) = sb1(s[1:i), s.i) \frown sb1(s[i+1:|s|], s.i),$$

meaning that s.i is removed from s, elements of s less than s.i are left unchanged, and elements bigger than s.i are subtracted by 1. Here, t=sb1(s, m), with m a value not occurring at s, is equivalent to the formal expression

$$(\forall k : 1 \le k \le |s| : (s.k < m \wedge t.k = s.k) \vee (s.k > m \wedge t.k = s.k-1)).$$

To remove an element of a permutation p of the elements of [n] in position i implies to shift, one position to the left, the elements to the right of the element removed, as well as subtracting 1 from all the elements bigger than it. This way, we bijectively obtain a permutation of [n−1].

Expressing P_n as $(\#p :: p \in pm[n])$, it is easy to calculate. Clearly P1 = 1. For n>1 we have

$$\begin{array}{ll} & P_n \\ = & \langle \text{ definition of } P \rangle \\ & (\#p :: p \in pm(n)) \\ = & \langle \textit{split} \text{ considering the value of } p.n \rangle \\ & (\Sigma i : i \in [n] : (\#p \in pm(n) :: p.n = i)) \\ & \langle \textit{map } p \textit{ to } t \text{ where } t = nrm(p,i) \rangle \end{array}$$

$$\begin{array}{ll} & (\Sigma i : i \in [n] : (\#t :: t \in pm(n-1))) \\ = & \langle \text{ definition of } P \rangle \\ & (\Sigma i : i \in [n] : P_{n-1}) \\ = & \langle \text{ constant term } \rangle \\ & n \cdot P_{n-1}. \end{array}$$

Observing the obtained recursive equation, we conclude that $P_n = n!$ for $n>1$. We have proved the following proposition.

Proposition 1: $(\#p :: p \in pm[n]) = n!, \quad \text{for } n \ge 1.$

B. Binomial Coefficients

We define the arithmetic function $\binom{n}{k}$ as the only solution to the functional equation on X a

$$X(n,k) = \begin{cases} 0, & \text{if } k>n \\ 1, & \text{if } k=0 \vee k=n \\ X(n-1,k) + X(n-1,k-1), & \text{if } 0<k<n \end{cases} \quad (0)$$

It is not hard to prove that n k corresponds to the number of subsets of size k, of a set of n elements (for instance [n], the set of integers from 1 to n). This shows the existence of a solution, its uniqueness is easily proved by induction.

Proposition 2: $\binom{n}{k} = (\#s \in set[n] :: |s|=k).$

Proof : Let $C(n,k) = (\#s \in set[n] :: |s|=k)$. Considering the cases defining equation (0), we note that the cases k >n, k= 0 and k=n are pretty simple, since in those cases, there is just one subset or none at a

We just study the case $0<k<n$:

$$
\begin{array}{ll}
& C(n,k) \\
= & \langle \text{ by definition } \rangle \\
& (\#s \in set[n] :: |s|=k) \\
= & \langle \text{ considering cases } n \in s \text{ and } n \notin s \rangle \\
& (\#s \in set[n] : n \in s : |s|=k) + \\
& \qquad (\#s \in set[n] : n \notin s : |s|=k) \\
= & \langle \text{ map } s \text{ to } c = s-\{n\} \rangle \\
& (\#c \in set[n-1] :: |c| = k-1) + (\#s \in set[n] : n \notin s : |s|=k) \\
= & \langle \text{ definition } ; n \notin s \wedge s \in set[n] \equiv s \in set[n-1] \rangle \\
& C(n-1,k-1) + (\#s \in set[n-1] :: |s|=k) \\
= & \langle \text{ by definition of } C \rangle \\
& C(n-1,k-1) + C(n-1,k)
\end{array}
$$

This proves that $C(n,k)$ is a solution for X in equation (0), and therefore, $\binom{n}{k} = C(n,k)$.

The two previous propositions show how to formally obtain a recursive definition for the solution of a counting problem. Next proposition correspond to Identity

Proposition 3: $(\Sigma k \in [0:n] :: (-1)^k \cdot \binom{n}{k}) = 0$

Proof:

$$
\begin{array}{ll}
& (\Sigma k \in [0:n] :: (-1)^k \cdot \binom{n}{k}) \\
= & \langle \textit{splitting} \text{ into even and odd cases for } k \rangle \\
& (\Sigma k \in [0:n] : even.k : \binom{n}{k}) - (\Sigma k \in [0:n] : odd.k : \binom{n}{k}) \\
= & \langle \text{ Proposition 2 } \rangle \\
& (\Sigma k \in [0:n] : even.k : (\#s \in set[0:n] :: |s|=k)) - \\
& \qquad (\Sigma k \in [0:n] : odd.k : (\#s \in set[n] :: |s|=k)) \\
= & \langle \text{ nesting ; counting operation } \rangle \\
& (\Sigma s \in set[n] :: (\#k \in [0:n] : k=|s| : even.k)) - \\
& \qquad (\Sigma s \in set[n] :: (\#k \in [0:n] : k=|s| : odd.k)) \\
= & \langle \text{ one-point rule } \rangle \\
& (\#s \in set[n] :: even.|s|) - (\#s \in set[n] :: odd.|s|) \\
= & \langle \textit{map } s \text{ to } t=s-\{n\}, \text{ if } n \in s;\ t=s \cup \{n\}, \text{ otherwise } \rangle \\
& (\#t \in set[n] :: odd.|t|) - (\#s \in set[n] :: odd.|s|) \\
= & \langle \text{ index renaming } \rangle \\
& 0
\end{array}
$$

CONCLUSION

This study was to demonstrate how to give the combinatorial arguments the formal content which we believe to have achieved. We did this to find ways to facilitate their automatic testing. We do not propose this formalization to replace informal arguments as alternative evidence. We believe, in fact, that the formal evidence can be read easily only by taking into account the informal ideas behind its original counterparts.

REFERENCES

[1] Achterberg, T. (2016): SCIP: Solving constraint integer programs. Mathematical Programming Computation 1, pp. 1–41.

[2] Aggoun, A., Beldiceanu, N. (2014): Extending CHIP in order to solve complex scheduling and placement problems. Mathematical and Computer Modelling 17, pp. 57–73

[3] Ahuja, R.K., Magnanti, T.L., Orlin, J.B. (2017): Linear Programming and Network Flows, 3rd ed. Prentice-Hall, Upper Saddle River, NJ

[4] Althaus, E., Bockmayr, A., Elf, M., Kasper, T., J¨uenger, M., Mehlhorn, K. (2012): SCIL— Symbolic constraints in integer linear programming. In: 10th European symposium on Algorithms, Lecture Notes in Computer Science, vol. 2461, pp. 75–87. Springer

[5] Andersen, H.R. (2017): An introduction to binary decision diagrams. Lecture notes, available online, IT University of Copenhagen

[6] Andersen, H.R., Hadˇzi´c, T., Hooker, J.N., Tiedemann, P. (2017): A constraint store based on multivalued decision diagrams. In: C. Bessiere (ed.) Principles and Practice of Constraint Programming (CP 2017), Lecture Notes in Computer Science, vol. 4741, pp. 118–132. Springer

[7] Appa, G., Magos, D., Mourtos, I. (2015): A polyhedral approach to the alldifferent system. Mathematical Programming 124, pp. 1–52

[8] Appa, G., Mourtos, I., Magos, D. (2012): Integrating constraint and integer programming for the orthogonal Latin squares problem. In: P. Van Hentenryck (ed.) Principles and Practice of Constraint Programming (CP 2012), Lecture Notes in Computer Science, vol. 2470, pp. 17–32. Springer

[9] Aron, I., Hooker, J.N., Yunes, T.H. (2014): SIMPL: A system for integrating optimization techniques. In: J.C. R´egin, M. Rueher (eds.) CPAIOR Proceedings, Lecture Notes in Computer Science, vol. 3011, pp. 21–36. Springer .

[10] Bacchus, F., Dalmao, S., Pitassi, T. (2014): Relaxation search: A simple way of managing optional clauses. In: AAAI Conference on Artificial Intelligence, pp. 835–841

[11] Bajestani, M.A., Beck, J.C. (2016): Scheduling a dynamic aircraft repair shop with limited repair resources. Journal of Artificial Intelligence Research 47, pp. 35–70

[12] Bajgiran, O., Cire, A., Rousseau, L.M. (2017): A first look at picking dual variables for maximizing reduced-cost based fixing. In: M. Lombardi, D. Salvagnin (eds.) CPAIOR Proceedings, Lecture Notes in

Computer Science, vol. 10335, pp. 221–228. Springer

Corresponding Author

Mr. Akash Pandey*

Research Scholar, Himalayan University Itanagar, Arunachal Pradesh