

Using Evolutionary Techniques to Hunt for Snakes and Coils

Abstract

The snake-in-the-box problem is a difficult problem in mathematics and computer science that deals with finding the longest-possible constrained path that can be formed by following the edges of a multi-dimensional hypercube. This problem was first described by Kautz in the late 1950's (Kautz 1958). Snake-in-the-box codes, or 'snakes,' are the node or transition sequences of constrained open paths through an n-dimensional hypercube. Coil-in-the-box codes, or 'coils,' are the node or transition sequences of constrained closed paths, or cycles, through an n-dimensional hypercube. Snakes and coils have many applications in electrical engineering, coding theory, and computer network topologies. Generally, the longer the snake or coil for a given dimension, the more useful it is in these applications (Klee 1970). By applying a relatively recent evolutionary search algorithm known as a population-based stochastic hill-climber, new lower bounds were achieved for (1) the longest-known snake in each of the dimensions nine through twelve and (2) the longest-known coil in each of the dimensions nine through eleven.

Introduction

Hunting for 'snakes' and 'coils' in an n-dimensional hypercube deals with finding the longest path, following the edges of the hypercube, that obeys certain constraints. First, every two nodes, or vertices, that are consecutive in the path must be adjacent to each other in the hypercube, and second, every two nodes that are not consecutive in the path must not be adjacent to each other in the hypercube. A third constraint, whether the path is open or closed, determines if the path is a snake or a coil. While coils have received the most attention in the literature (Harary, Hayes, and Wu 1988), both snakes and coils have useful applications. This paper details our attempt to hunt for both snakes and coils using evolutionary search algorithms.

An n-dimensional hypercube contains 2^n nodes that can be represented by the 2^n n-tuples of binary digits of the hypercube's Gray code as illustrated in Figures 1 and 2. By labeling each node of the hypercube with its Gray code representation, adjacencies between nodes can be detected by the fact that their binary representations differ by exactly one coordinate (Harary, Hayes, and Wu 1988). Using this fact, one can immediately detect if any two nodes in the hypercube are adjacent by performing the logical exclusive-or, hereafter referred to as XOR, operation on them and confirming that the result is an integer power of two. Using this information, a strategy for detecting, as well as generating, node adjacencies can be generalized to any dimension.

In Figure 1, the bold line segment illustrates an open path, or snake, through a three-dimensional hypercube. This path, {0, 1, 3, 7, 6} in integer representation and {000, 001, 011, 111, 110} in binary representation, is one specific example of the longest-possible snake for a three-dimensional hypercube. The length of this snake is four, as the length of a snake always refers to the number of transitions, or edges, in the snake.

In Figure 2, the bold line segment illustrates a closed path, or coil, through a three-dimensional hypercube. This path, {0, 1, 3, 7, 6, 4, 0} in integer representation and {000, 001, 011, 111, 110, 100, 000} in binary representation, is one specific example of the longest-possible coil for a three-dimensional hypercube. The length of this coil is six, as the length of a coil always refers to the number of transitions, or edges, in the coil.

Each of these paths can also be represented in the form of a transition sequence, such as {0, 1, 2, 0} for the snake and {0, 1, 2, 0, 1, 2} for the coil. Here each number in the sequence is the coordinate of the Gray code bit which is being changed to generate the next node of the snake or coil. Transition sequence representations can start from

either zero or one, but in this paper we follow the convention of starting with zero as it both seems more intuitive and simplifies the mathematics when converting between node sequences and transition sequences. All snakes and coils listed in this paper will follow the form of transition sequence representation in order to conserve as much space as possible.

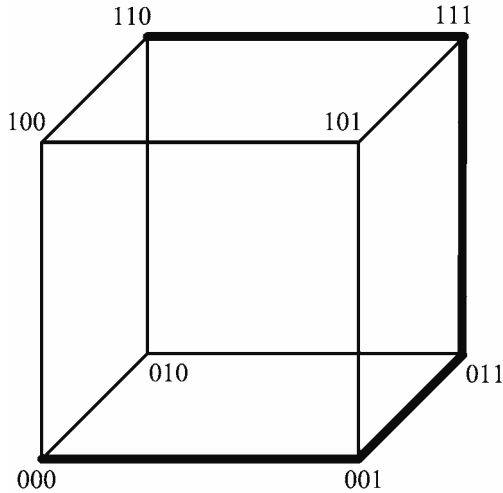


Figure 1: 3-D hypercube with embedded snake.

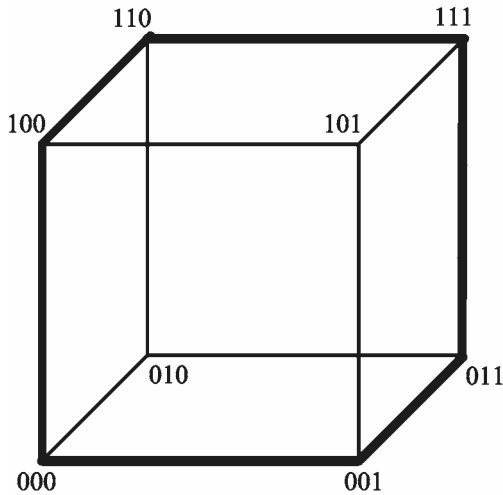


Figure 2: 3-D hypercube with embedded coil.

Previous Approaches to Hunting for Snakes and Coils

Traditionally, mathematical approaches to the snake-in-the-box, as well as the coil-in-the-box problems, hereafter referred to collectively as the path-in-the-box, or PIB, problem have involved two fundamental strategies. The first is a method of construction utilizing the tools of logic, discrete mathematics, and graph theory, while the second uses mathematical analysis to reduce the search-space followed by a computationally exhaustive search of the remaining, somewhat more-tractable, search-space. These techniques have been successful in determining the longest-possible snakes and coils for hypercubes of dimensions one through six as shown in Table 1. For dimension seven, the lower bound for longest-possible snake (Potter et. al. 1994) was found independently using both a genetic algorithm and an exhaustive search, while the lower bound for longest-possible coil (Kochut 1996) was found using only an exhaustive search.

DIM	SNAKE	COIL
1	1	0
2	2	4
3	4	6
4	7	8
5	13	14
6	26	26
7	50	48

Table 1: Maximum lengths of snakes and coils.

But even as mathematical approaches to solving this problem have been strengthened through the use of computational techniques, the combinatorial explosion in the size of the search-space as the dimension number increases has become a barrier for these methods. For dimensions eight and above, even with currently available hardware, an exhaustive search remains impractical. This has opened the door for less-traditional, heuristic-based computational search techniques. One increasingly popular branch of these search techniques is known as *stochastic search algorithms*. This area includes stochastic hill-climbers, tabu search, simulated annealing, evolutionary strategies, genetic algorithms, and hybrids of these particular types such as memetic algorithms.

One example of a stochastic search algorithm that has been used to hunt for snakes and coils in dimensions seven and eight is the genetic algorithm, hereafter referred to as

the GA. Developed by John Holland (Holland 1975), the GA is based on the simulation of Darwinian evolution and uses an evolutionary loop composed of fitness-based selection of individuals from a population, crossover of the individuals' genetic material, and mutation of the individuals' genes. The GA performs a search-space reduction through the use of a heuristic in determining the fitness of an individual within the population and through the inherent parallelism of a population-based approach. This technique has met with success and, by finding, what were at that time, record-breaking snakes in dimension seven and eight (Potter et. al. 1994), proved its effectiveness for snake hunting.

Another stochastic search algorithm that has proven successful in traveling salesperson-type problems, such as the PIB problem, is the stochastic hill-climber (Kingdon and Dekker 1995). A population-based stochastic hill-climber, hereafter referred to as PBSHC, is very similar in structure and operation to the simple GA with a few key differences in the evolutionary cycle. The first difference is the absence of a crossover operator. Where the GA models sexual reproduction, the PBSHC models asexual reproduction in that the children in each generation are created directly from the parents of the previous generation without the exchange of genetic material between them. The second difference is the addition of a growth operator. The growth operator is the component that does most of the actual hill-climbing as it chooses randomly from the nodes available to extend each snake's path by one edge along the hypercube. The absence of crossover, broadly considered the most important operator of the GA, can sometimes leave the PBSHC at a relative disadvantage. However, due to the trouble most crossover schemes have in dealing with global adjacencies, the lack of crossover did not seriously degrade the PBSHC's performance, relative to the GA, in the case of the PIB problem.

The PBSHC Evolutionary Cycle

Each individual in the population consists of a sequence of integers that represents the node sequence of a snake, or valid path through the hypercube, in the dimension being searched. These individuals are initialized as either a snake of length zero, that is consisting of only the zero node, or seeded with a pre-existing snake of choice. Following initialization, the evolutionary cycle begins its first generation. Each generation begins with a fitness evaluation. The fitness function used is based on both the length and the 'tightness' of the snake. The tightness of a snake is a measure of how many nodes are left available in the hypercube after subtracting all those nodes that are disqualified either by already being in the snake or by

being adjacent to a node, other than the end node, that is already in the snake. The choice of tightness as a component of the fitness function was inspired by the idea that tighter snakes might tend to be longer snakes.

Initial attempts of integrating both length and tightness into a fitness function met with limited success. Many combinations of linear and non-linear factors of length and tightness for the fitness function were tried. While they each would perform well in some periods of evolution, they would inevitably perform worse in others. Once the average fitness of the population slowed, the diversity of the population would fall and the fitness function would converge to a local optimum. Our first attempt at solving this problem was to develop an adaptive fitness function that would balance between the importance of tightness and length, using the diversity of the population as the balancing factor. This technique also failed, reacting to changes in diversity too quickly in some cases and not quickly enough in others. Our second attempt was a more simplistic compromise between the two objectives. Length was set as the overriding objective within each generation, and tightness delegated to an ordered ranking factor of the rank-based selection within each generation. Since all snakes that were able to grow in the previous generation have the same length in the current generation, tightness becomes the only distinguishing factor of the fitness function.

After the individual fitness of each of the snakes in the population is determined, the population is subjected to selection based upon each snake's fitness. Three fundamental selection types were considered. Preliminary trials were conducted using roulette-wheel, tournament, and rank-based selection methods. Roulette-wheel, or probabilistic selection, proved the least effective, and most computationally expensive. Tournament selection was more effective than roulette-wheel at producing longer snakes, on average, over multiple runs. Rank-based selection out-performed both roulette-wheel and tournament selection, by maintaining a more diverse population throughout the evolutionary cycle. In rank-based selection, after first ranking the population by fitness, a predetermined percentage of the population is selected to carry over to the next generation. In order to keep the population constant throughout the entire run, the remaining places available in the population of the next generation are filled starting again with the most fit individuals. For example, if the selection percentage is 90%, the individuals are ranked by fitness and the first 90% are selected. This still leaves 10% of the next generation empty, so the first 10% from the current generation are used to finish the selection operation.

After selection, the growth operator grows each snake in the population by one step each generation, connecting

each snake's end node to one of its adjacent nodes that has not been disqualified by already being in the snake, or by being adjacent to some node that is in the snake. A modification was made to the growth operator in order to grow coils as well as open-path snakes. This modification allows for the selection of end nodes that are adjacent to the current snake's start node, resulting in the formation of a coil. When this happens, the coil can no longer be extended under this algorithm. In order to keep the entire population eligible for extension, the fitness function was also modified to record any coils found, and then replace them with snakes of zero fitness that will be replaced in the next generation. The result of these modifications is that the same effective operators could be used to hunt for both snakes and coils, simultaneously. While bi-directional growth was used during preliminary trials in dimension eight, unidirectional growth was chosen for this implementation to best allocate computational resources. This operator can be seen to perform a stochastic hill-climbing process on each snake in the population as the choice of which adjacent node to connect to is based on random selection from the available nodes. A choice was made early to grow all snakes in the population instead of only the snake of best fitness (note: typically, enhancing the best individual in a population is a standard approach used in hybrid genetic algorithms (Potter et. al. 1992)). This choice was also based on results of a comparison of these two approaches in an earlier GA implementation. Growing all the individuals within the population also works well in conjunction with the fitness function which requires that all snakes in the population of a given generation be of the same length in order to function properly. The fitness function consists of the sum of the snake's length and normalized tightness. This results in the fitness function simplifying to a function of tightness alone as the length component of the fitness value will dominate for snakes of different lengths, yet cancel for snakes of the same length. This also results in the automatic elimination of snakes that can no longer grow, allowing their place in the population to be reallocated to other snakes that are still capable of growth.

After growth, the mutation operator acts on each snake in the population. The mutation operator we experimented with is an enhancement of a basic XOR mutation scheme (Brown 2004). In the standard XOR mutation scheme, a node is chosen at random from within the snake, excluding the start node and the end node. The chosen node's neighboring nodes are logically XOR'd and that result is then XOR'd with the chosen node in order to exchange it with a different node that maintains local adjacency requirements with the original node's neighbors. The enhancement of this operator is referred to as iterative-conditional XOR mutation and is somewhat more computationally expensive, but also more effective. Instead of choosing a node at random, each node in the

snake, with the exception of the start node and the end node, is mutated and tested for any improvement in fitness within a copy of the original snake. If any improvement is found, that mutation is added to a mutation pool. Upon testing of all nodes within the snake, the snake of best fitness from the mutation pool is substituted for the original snake. If any improvement was found through mutation, the entire process is repeated until no further improvement is found. This scheme ensures that only constructive mutation is allowed, and that undergoing mutation can never reduce an individual's fitness. By making this mutation conditional, it has also become a hill-climbing component of the evolutionary cycle. However, left to iterate without bounds this scheme may lead to prohibitive runtimes. Restricting the number of iterations in this scheme to five or less keeps the advantages of a local hill-climber and also keeps the runtime under control. It turns out that this enhancement did not influence our results because the operator was turned off in order to prevent changes in the high-quality root-snakes used when seeding results from lower to higher dimensions.

Hunting for coils is a much more difficult problem than hunting for snakes due to the added constraint of having to be a closed path. Two approaches were attempted in the effort to evolve populations of coils. The first approach was made using an operator that allowed each coil in the population to replace a random node with three nodes that maintained a valid coil while becoming two edges longer in the process. A second approach used the snake-hunting algorithm and then allowed the snakes to become coils by permitting the snakes to grow to nodes that were adjacent to the start node, recorded the resulting coils, and then removed them from the population. The second approach, which proved to be much more effective, was subsequently chosen as the method we used to hunt for coils in dimensions nine through twelve.

The PBSHC Experimental Setup

The choice of parameter settings was found to be of key importance to the performance of the PBSHC and these settings were tuned extensively in dimension eight before modified to run in dimensions nine through twelve. Our previous experience with genetic algorithm snake hunters supported the application of lower-dimension parameter settings as a baseline for higher-dimension runs. The fitness function is set to the sum of length and normalized tightness. Normalized tightness was defined as the number of nodes remaining available divided by the total number of nodes in the n-dimensional hypercube. Rank-based selection was chosen in order to maximize diversity within the population. While both unidirectional and bi-

directional growth were used in the dimension-eight trials, the relatively memory-conservative, unidirectional implementation was chosen for the higher-dimensional runs in order to maximize potential population sizes for dimensions nine through twelve. Because this particular implementation's chromosome size is constant, the use of unidirectional growth instead of bi-directional growth allowed the required memory for each population size to be reduced by half. Population sizes from one hundred through ten thousand were run in trials using mutation. Larger populations were prohibitively time-consuming using mutation, especially for dimensions eleven and twelve.

Results

The best results to date were achieved using populations of ten thousand, a selection percentage of ninety percent, and by seeding each population with highly-fit, lower-dimension snakes at startup. Using a technique where the best snakes found in each dimension were used as seeds for the next higher dimension (Potter et. al. 1994), the PBSHC was able to find new lower bounds for snakes in dimensions nine through twelve and new lower bounds for coils in dimensions nine through eleven. In order to generate good seeds for dimension nine, a bootstrap solution was used. This method involved cutting a dimension-eight, length-97 snake (Rajan and Shende 1999) back to its length-50 root, corresponding to the longest snake in dimension seven, and running an exhaustive search on that snake to generate a pool of seventeen distinct length-97 snakes. These snakes were then used to seed the dimension-nine runs. Subsequently, the best snakes found in each dimension were used as seeds for the next dimension's runs. In conjunction with this technique, mutation was shut off in order to preserve the high-quality seeds generated from each previous dimension. This also reduced the runtime allowing larger populations to be run over a much shorter time-frame.

In Table 2, a comparison is made between the previously known lower bounds for snakes and the results achieved using the PBSHC, with new lower bounds in bold. As shown in Table 3, new lower bounds were also achieved when applying the PBSHC to the task of hunting for coils in dimensions nine through twelve. Although the PBSHC performed well in hunting for both snakes and coils, it failed to match or raise the lower bounds on coils for dimension eight or twelve, demonstrating that hunting for coils is an inherently more difficult problem than hunting for snakes.

Another factor to consider at this point is that by starting each run with lower-dimension seeds of high fitness means that the search space is restricted to branches having those seeds as their root, eliminating the huge number of potentially long snakes that do not extend from that branch.

DIM	PREVIOUS BEST	PBSHC
8	97	97
9	168	186
10	338	358
11	618	680
12	1236	1260

Table 2: Comparison of old and new lower bounds for snakes.

DIM	PREVIOUS BEST	PBSHC
8	96	90
9	170	180
10	340	344
11	620	630
12	1238	1236

Table 3: Comparison of old and new lower bounds for coils.

In Table 2, for dimension eight, the lower bound for longest-known snake (Rajan and Shende 1999) was found using traditional mathematical proof and construction techniques aided by computational search (i.e., construction based on lower dimension long snake characteristics). For dimensions nine through twelve, the previous best-known lower bounds were derived by calculation from the lower bounds for longest-known coils in each dimension. Any coil can be converted to a snake by removing one node. This results in a snake whose length is two less than the original coil, as removing one node removes the two edges connecting it within the coil. The previous best-known lower bounds were 168 for dimension nine (Abbott and Katchalski 1991), and 338, 618, and 1236 for dimensions ten through twelve, respectively (Paterson and Tuliani 1998). The transition sequences representing the new lower bounds for snakes in dimensions nine through twelve are listed below.

In Table 3, for dimension eight, the lower bound for longest-known coil (Abbott and Katchalski 1991) was found using traditional mathematical proof and construction techniques. The previous best-known lower bounds were 170 for dimension nine (Abbott and

Katchalski 1991), and 340, 620, and 1238 for dimensions ten through twelve, respectively (Paterson and Tuliani 1998). The transition sequences representing the new lower bounds for coils in dimensions nine through eleven are listed below.

New Lower Bound Snakes

dim 9 : 186

01234532103253123452310326054301
 35023103543012502372052413015201
 30543520362543013502310354301250
 28732453103453152103453013561301
 52013054352031762543513253013562
 54350135430520154103562350

dim 10 : 358

01234532103253123452310326054301
 35023103543012502372054213015201
 30543520362543013502310354301250
 28732453103453152103453013561301
 52013054352031762543513253013562
 54350135430520154103562350951078
 05210345301320531034526302534503
 10251031245081623013254351325301
 53603102503423012073421301520130
 54352036504532513502354082134503
 10235024523103620523142105203120
 542856

dim 11 : 680

01234532103253123452310326054301
 35023103543012502372054213015201
 30543520362543013502310354301250
 28732453103453152103453013561301
 52013054352031762543513253013562
 54350135430520154103562350951078
 05210345301320531034526302534503
 10251031245081623013254351325301
 53603102503423012073421301520130
 54352036504532513502354082134503
 10235024523103620523142105203120
 54285610879320521034530132053103
 45263025345031025103124508162301
 32543513253015360310250342301207
 34213015201305435203623103453152
 35431821345031023502452310362052
 31421052031205427915205430520362
 31025302407450235210523543084015
 20132630250342052680263025345031
 05210312437052061028620362543013

50231035431063201450310253725042
 305260342

dim 12 : 1260

01234532103253123452310326054301
 35023103543012502372054213015201
 30543520362543013502310354301250
 28732453103453152103453013561301
 52013054352031762543513253013562
 54350135430520154103562350935263
 02534503102510315620531523543104
 71352310235408310345301320531034
 53102372054203520130543520164102
 51352735103123513054352031284054
 32103243052014610236217581520351
 83109421301520130543520362543013
 50231035430125074103253123452310
 32605430135023103543012594130543
 52036254301350231035430128921034
 53013205310345263025345031025103
 12450960231025345031023562315234
 53013841301537253152015412530263
 25345031025103410814532513502354
 12903450310251031245231036205430
 52031205491450315321301490250912
 03620115094305106503450617402310
 35430125135430126103123513054352
 03128154305201561352310276201305
 43520362543013502610130250345026
 32534503153450740231035430120410
 34526302534503153213015310254615
 32013567249102130152013054352036
 23520345325135023571352054352012
 40235623152013410130253450310251
 03165310354301204103451823102534
 50310235623152031762054814502530
 13561325043501926513012509405865
 13054352036235203453010913205471
 45031023562502130250345026301351
 43164259413054352036235203453251
 35025613014852312503213108324613
 46051106327905210345625430541052
 10312450945214301310691456315410
 341243521080250342631034

New Lower Bound Coils

dim 9 : 180

01234532103253123452310326054301
 35023103543012502376321035430135
 02310354301362435103543124103548
 25430135023103543012563543213523

01235432134765025312502354321520
1320640231054301321

dim 10 : 344

01234532103253123452310326054301
35023103543012502372054213015201
30543520362543013502310354301250
28732402310354301251354301271031
23513054136501354305203120527532
01305410523103753014713490453013
57130125013054184502310250345315
21035730132543502354321345025306
50132054814502531250235785320521
35205435201320840231025034132103
50630137231620135815235
542856

dim 11 : 630

01234532103253123452310326054301
35023103543012502372054213015201
30543520362543013502310354301250
28732453103453152103453013561301
52013054352031762543513253013562
54350135430520154103562350951078
05210345301320531034526302534503
10251031245081623013254351325301
53603102503423012073421301520130
54352036504532513502354082134503
10235024523103620523142105203120
54285610435821523625435203625426
12436503402369102503453105345273
20132930250162109701234523103653
01381924305203623153295310354301
25135430165013054352019610236274
93013205279250490273942563502310
29032531234523103260315065816201
32542652017547190523103260310521
0368523152643240163809

Conclusions

The results of using a PBSHC to search for PIB codes are very encouraging. As computational hardware improves over time, this technique should prove useful in even higher dimensions. Further improvements using this implementation of the PBSHC are currently being considered. We anticipate that this technique would work well as a distributed process, with each thread searching a specific sub-region of the hypercube for a linear factor increase in search speed.

Acknowledgements

We would like to thank the students and staff of the AI Center, especially Dr. Michael Covington, for their support and cooperation with this project. There were numerous days and nights when the Center's lab computers were occupied with snake and coil hunting.

References

Abbott, H. L. and M. Katchalski. 1991. On the Construction of Snake-In-The-Box Codes. *Utilitas Mathematica* 40:97-116.

Brown, W. April, 2004. Personal Communication.

Harary, F., J. P. Hayes, and H. J. Wu. 1988. A Survey of the Theory of Hypercube Graphs. *Computational Mathematics Applications* 15:277-289.

Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: The University of Michigan Press.

Kautz, W. H. 1958. Unit-Distance Error-Checking Codes. *IRE Trans. Electronic Computers* 7:179-180.

Kingdon, J. and L. Dekker. 1995. The Shape of Space. In *Proceedings of the Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, 543-548. London, UK.: IEE.

Klee, V. 1970. What is the Maximum Length of a d-Dimensional Snake? *American Mathematics Monthly* 77:63-65.

Kochut, K. J. 1996. Snake-In-The-Box Codes for Dimension 7. *Journal of Combinatorial Mathematics and Combinatorial Computations* 20:175-185.

Paterson, K. G. and J. Tuliani. 1998. Some New Circuit Codes. *IEEE Transactions on Information Theory* 44(3):1305-1309.

Potter, W. D., J.A. Miller, B.E. Tonn, R.V. Gandham, and C.N. Lapena. 1992. Improving The Reliability of Heuristic Multiple Fault Diagnosis Via The Environmental Conditioning Operator. *APPLIED INTELLIGENCE: The International Journal of Artificial Intelligence, Neural Networks, and Complex Problem-Solving Technologies* 2: 5-23.

Potter, W. D., R. W. Robinson, J. A. Miller, and K. J. Kochut. 1994. Using the Genetic Algorithm to Find Snake-

In-The-Box Codes. In *7th International Conference On Industrial & Engineering Applications Of Artificial Intelligence and Expert Systems*, 421-426. Austin, Texas.

Rajan, D. S. and A. M. Shende. 1999. Maximal and Reversible Snakes in Hypercubes. Presented at the *24th Annual Australasian Conference on Combinatorial Mathematics and Combinatorial Computation*.