MASIB: Multi Agent Snakes In a Box

Yossi Cohen
Collaborating Multi-Agents Systems Project
Dept. of Software and Information Systems Engineering,
Ben-Gurion University of the Negev,
Beer-Sheva, Israel

Abstract

One of the main motivation for solving snake in the box problem is that the vertices sequence of a solution can be transformed to *Gray Code*. But what if we want to have more than one sequence for specific amount of bits? Different sequences can represent different information context with error detection. This work defines the problem and presents number of methods to solve the multi agent snake in the box problem using heuristic search.

Introduction

The Snake in the box problem (SIB) is a classic CS problem formalized in work from over half a century ago (Kautz 1958). SIB solution can be presented as gray code sequences that are being able to detect any single bit error (or even more, depends on the *spread* of the snake - will be explained later more deeply) making it important to solve. on the other hand, SIB suffers from an exponential explosion - therefore it is being investigated by many researchers from different CS fields using methods such as exhaustive search, genetic algorithms, Monte Carlo and more. as mentioned above, the result of SIB can be translated to Gray Code sequence that allows error detection, We would like to offer a new aspect to this problem: seeing it as a multi agent (agents will be named as snakes) problem - i.e. two snakes. A solution to the multi agent problem can be used as separate sequences allowing to distinguish between the different sequences and allow error correction for each sequence.

There are some intuitive methods that one can think of for generation of SIB separate sequences for instance, if you want 2 separate snakes over 8 dimensions (bits) you can use the first 4 bits for the first snake and the next 4 bits for the second. or just solve the problem as single snake and split it. in this paper we will discuss the different methods, define and offer solutions to the Multi-Agent Snake In the Box problem (MASIB).

Single snake problem definition SIB problem is defined over a graph $Qn = (Vn, En)$ The graph represents an hypercube with n dimensions therefore $|Vn| = 2^n$ Every vertex in the graph is represented by bit vector and have edges to all vertices with *Hamming distance* equals 1 hence only one bit difference from it. The first vertex in the path denoted $tail(s)$ and the last $head(s)$. the last property is the snake's *Spread*; a snake with spread k cannot pass through vertices that are fewer than k hops away from other vertices in it. If the hamming distance is smaller than k then the vertices must be with distance less then k in the snake vertices order. It is easy to see that the bigger the k is the smaller the snake can be, but although bigger k means smaller snake in some application bigger k means better error detection and therefore desirable. (n, k)SIB is the search problem of finding the longest snake in Qn with a spread of k (Palombo et al. 2015).

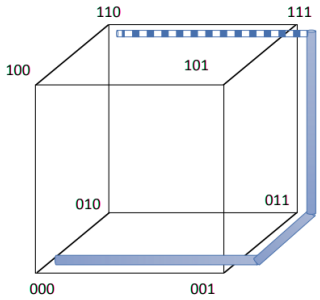

Figure 1: An example of a snake in a (3,2)SIB problem: 3D cube with *snake* on the edges

Example: Consider snake in Figure 1, it is consist of the sequence of vertices: $s = (000, 001, 011, 111)$. $tail(s) = V(000)$ and $head(s) = V(111)$. Note that $V(101)$ cannot be appended to s as the next $head(s)$, since it is adjacent to $V(001)$ that is a part of s. By contrast, the vertex $V(110)$ can be added to s since it is not adjacent to any other vertex in s. adding $v(110)$ to the snake is a feasible result to the $(3, 2)$SIB problem. You can also see that the example above $(s = (000, 001, 011, 111, 110))$ does not have spread of $k = 3$ ($(3, 3)$SIB) since $Hamming(000, 110) = 2$ but $V(110)$ is more than k in the snake order ($v(000)$ index is 0 and $v(110)$ index is 5).

Multiple snakes problem definition MASIB problem is defined as follows: (n, a, k, l)MASIB is the search problem of finding the *maximal* sum of snakes lengths $a = (s_1, s_2...s_a)$ in Qn with an intra-snake spread of k and inter-snake spread of l. that means the Hamming distance of a snake from itself can't be lower than k as in SIB problem and the Hamming distance between any part of two different snakes can be at most l. The *maximal* property of this problem is w.r.t the sum of all a snakes lengths later on we will add constraint on this property to allow only similar length snakes in order to get more practical and less complex solution space. notice that if there is only one snake, the inter-snake spread(l) is irrelevant and we get a simple SIB problem, or in more formal way: $(n, 1, k, *)MASIB \equiv (n, k)SIB$ Notice that when 2 snakes collide and can't grow anymore there can be a situation were there is still vertices with legal spread *k/l* we will call them *Free Spots* and try to understand their nature empirically.

Research Questions The main questions we are asking in this paper are:

- Does solving MASIB as a multi agent problem has any advantages over splitting a single snake into some pieces?
- Does optimal solution for SIB and MASIB must be free of *free spots*?
- What is the preferable method to solve this problem?

Since this problem is quite new, we will firstly define it than offer some solution based on heuristic search methods for multi agent problems.

Related Work

Previous work (Palombo et al. 2015) offers methods to solve the SIB problem with heuristic search techniques, SIB is a Longest Path Problem (LPP) with additional constrains. The goal is to find the implicit longest path over a graph that represents a N-dimensional hypercube. Another constraint is that the path must be induced: path can't have any two vertices which are adjacent but are not part of the path. The paper suggests several heuristics for solving SIB and CIB (we will not cover to the Coil variant of the SIB problem in this work) as well as definition of the problem as a search problem, the search definition goes as follows: A node represents a snake $s = (v_0, v_1, ...v_j)$, a snake is valid only if it has a spread of k. Expanding a node means generating all nodes that represent valid snakes obtained by appending a single vertex to the head of the snake. A goal node is a snake to which no vertex can be appended (all vertices are valid for goal but if we can expand a node then there is exist a longer snake thus we are not in the goal yet). The reward of a node is the number of edges in the snake, and the task is to find the longest snake. Heuristic must be admissible for MAX problems - upper bound for the solution. In SIB $g(s)$ is the length of the snake and $h(s)$ is an upper bound on the length of the snake that starts from $head(s)$ and together with s form the longest valid snake. (Palombo et al. 2015) provide numerous heuristics with different levels of efficiency and complexity for this problem. in this work we implemented the $legal$ and $reachable$ heuristics offered there.

Since we are solving this problem for multiple snakes we need to use methods form the multi agent search field. one of the most basic operation we can do when want to expand a multi agent search node is a Cartesian product of all possible moves (Wang and Botea 2008) of all agent (includes the $wait$ move) in our case, that means that when we wants to expand a node we will calculate each snake possible moves w.r.t k spread and generate all the possible combinations of legal moves with l spread, thats can be very costly in the aspect of branching factor. consider $n = 7$ and $|a| = 3$ we can get $7^3 = 343$ generated nodes when expanding a single node.

To overcome the huge branching factor issue the Operator Decomposition method was offered by (Standley 2010) according to this method we are not expanding all the agents combinations together as a Cartesian product of the agents' moves, instead, every time we need to expand a node we will expand only one agent and in the next level we will expand the next agent in a round-robin approach. In more formal way we say that on any level L in the search tree we will expand only the agent with index number: L modulo $|a|$. Operator Decomposition will assist us to reduce the branching factors - but there are no free meals, the search tree depth will increase when using this method.

Methods

This section elaborates on the possible ways to generate separated multiple induced sequences. The first subsection details about splitting the hypercubes into distinct hypercubes. The next explains about splitting one sequence into multiple sequences and the last details the aspects of solving this problem as a multi agent search problem.

Splitting the hypercube bits With minor changes to the above definition of MASIB (i.e. ignore the inter-snake spread l) one can decide to split the entire hypercube into $|a|$ separate hypercubes and solve every snake in it's own cube. One will need to solve the issue of distinguish between the sequences since every time we transmit n bits the n bits will hold the representation of all agents. We will reject this approach since it produce poor results. This can understand with a simple example, Consider the two sequences (decimal representation) $a = [s_1 = (0, 1, 3, 7, 15, 31, 63, 62, 60), s_1 = (202, 234, 232, 233, 249, 241, 245, 213, 212)]$ this two sequences are valid $(8, 2, 4, 4)$MASIB solution with $g(a) = 16$ and every sequence have $g(s_i) = 8$. now lets assume we want to split it into two hypercubes with Q_4 each. you can see on Table 1 that $(4, 4)$SIB solution has $g(4)$ and in the above example we shown two sequences with $g(s_i) = 8$ that is better than any of $Q4$ possible k values. Because of that, we will not try to develop this approach any more in this paper.

Cutting snakes Another method one can choose is to take an existing solution of SIB and split it into separate snakes.

n \ k	2	3	4	5	6	7
3	4	3	3	3	3	3
4	7	5	4	4	4	4
5	13	7	6	5	5	5
6	26	13	8	7	6	6
7	50	21	11	9	8	7
8	98	35	19	11	10	9
9	?	?	28	19	12	11

Table 1: Optimal solution for different (n,k)SIB, source:(Palombo et al. 2015)

the benefit of this method is that we don't need to solve anything - there are existing published sequences of different (n,k)SIB that we can use, in the next section we will see that when we split a snake we get a shorter solution then if we decide to generate the snakes separately, moreover, if we split an existing (n,k)SIB sequence we can get only (n,a,k,l)MASIB with $l \leqslant k$ since the original sequence had intra-snake spread of k that's means we can't guarantee bigger inter-snake spread because the original sequence don't hold this property. in order to allow different spreads we must search the solution for the specific l,k spreads.

Multi agent search problem In this section we will refer to the different aspects of the MASIB search problem and explain how we decided to refer them.

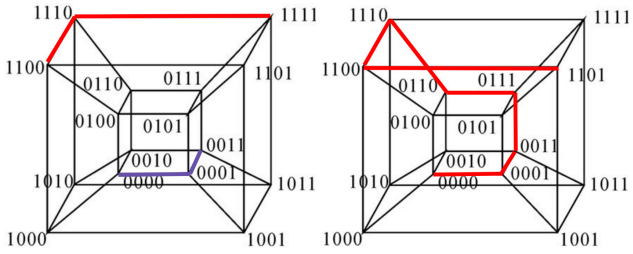

Figure 2: Single snake vs. two snakes with the same spread

Growing Snakes In order to solve the (n,a,k,l)MASIB problem we need to decide where to grow the snakes *from*. On SIB problem this is not an issue at all since the domain is symmetric and we can start always from 0. In contrast, on MASIB we can set the starting location of one snake to 0 (For convenience - it will be the first agent) but the other snakes starting location have immediate affect on the search result. If we will decide to grow two snakes with initial location that has Hamming distance that is less than l we will not need to expand any node since the solution always will be *illegal*. Another example: Consider the two hypercubes in Figure 2, the right hypercube shows a solution for the $(4,2)$SIB and the left shows the solution for the $(4,2,2,2)$MASIB it is trivial to see that adding snakes to the cube decreasing significantly the accumulated snakes lengths. What is less trivial is that the on MASIB agents' starting location will influence the maximal possible sum of snakes length. Consider Figure 2 again, on the right side of the figure you can see a single $(4,2)$SIB with $tail(s) = V(0000)$ and $head(s) = V(1101)$ and total length: $g(s) = 7$, if we will split it to two snake with $l = 2$ (inter snake spread) between them we can get two sequences: $a = [s_1 = (0000, 0001, 0011, 0111), s_2 = (1110, 1100, 1101)]$ that are legal $(4,2,2,2)$MASIB solution with $g(a) = 5$, but if we will *choose* to grow two snakes starting from $[tail(s_1) = V(0000), tail(s_2) = V(1111)]$ we will get the result on the left: $a = [s_1 = (0000, 0001, 0011), s_2 = (1111, 1110, 1100)]$ with $g(a) = 4$ which is optimal for those specific starting points but not the best solution for $(4,2,2,2)$MASIB. On Figure 3 right side you can see another solution for

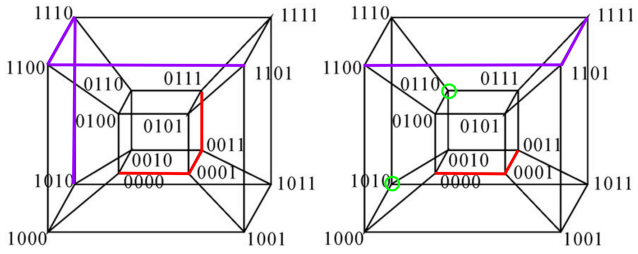

Figure 3: On the left - optimal solution for $(4,2,2,2) - MASIB$. On the right, example for unreachable free-spots.

$(4,2,2,2)$MASIB with same starting points as in Figure 2 right side $[tail(s_1) = V(0000), tail(s_2) = V(1111)]$ and the same $g(a)$ value. But on this solution $(a = [s_1 = (0000, 0001, 0011), s_2 = (1111, 1101, 1100)])$ there are two locations $V(0110), V(1010)$ that far enough from both snakes but on the same time unreachable for them both, we will refer to those locations as *free spots* and explain more about them later on. Lastly we can see on the right side of Figure 3 the optimal solution for $(4,2,2,2)$MASIB with $g(a) = 6$ $(a = [s_1 = (0000, 0001, 0011, 0111), s_2 = (1101, 1100, 1110, 1010)])$ notice that we can get longer accumulated sequences (bigger g value) when we grow the snakes *separately* - This means that we have now a clue for our first research question - solving the MASIB as Multi Agent problem can give us better results but since we don't know what initial vertex to grow the snake **from** that means we have to grow from every possible location.

If we consider symmetry, on 2 snakes MASIB, we can start the first snake always from 0 but the second snake must be start from approximately 2^n locations, (we can ignore all illegal starting locations such as initial locations with spread smaller than l). In more general manner, for more than two agents we will get *approximately* $\binom{2^n}{|a|-1}$ starting points - to start grow snakes we assume $|a|-1$ because we can set the starting position of the first snake always to 0, and we stated *approximately* again because of the illegal initial states.

Another important thing to take into account when growing multiple snakes is keeping the different spreads correct.

we suggest doing it as follows, first we need to generate all the child nodes of every snake as doing in SIB - alternate a single bit every time, when doing this we will validate that all those nodes have legal spread of k, than, in order to suggest next node for the MASIB we will offer only variation of all the a snakes that have valid l spread. Since every time we grow the snakes by generating only the valid nodes on the intra and inter level we assure that all the nodes in the open list will be valid.

Free Spots When solving multi agents problems sometimes one agent can get in the way of another. This problem significantly exists in the MASIB domain since that even a single snake can prevent itself from growing with it's own sequence (body). Consider Figure 2 right side, There is a solution for $(4, 2, 2, 2)$MASIB there when starting locations of the snakes are $[tail(s_1) = V(0000), tail(s_2) = V(1111)]$ both snakes have been grown to the point where they cannot grow anymore so we have the sequences $[s_1 = (0000, 0001, 0011), s_2 = (1111, 1101, 1100)]$ and $g(s_{\text{a}}) = 4$ but thought they can't grow there are still places in Qn with Hamming distance less than k and l. We will call this places *Free Spots*. notice that free spots can be calculated w.r.t the intra-snake spread or the inter-snakes spread. a rule of thumb can be that if we have free spots on our solution maybe we need to stop current search and look for a better solution. Does this intuition correct? on this work we will summary the free spots of any solution and try claim basic claim according to the empiric results.

Snakes size The reader can easily notice that solving MASIB can be much more complicated than SIB. Consider the MASIB for two snakes for instance, you can grow only the first snake leaving the second snake with $g(s_2) = 0$ another option is to grow the first snake with second snake $g(s_2) = 1$, the longer the second snake the shorter the first snake can be until a solution of first snake $g(s_1) = 0$ and grow only the second. In order to prevent this we suggest a constraint on the all snakes sizes: *A solution is valid iff the maximal snake length is greater then the minimal snake length by* 1 *at most* therefore $Max(\forall g(s_{\text{i}})) - Min(\forall g(s_{\text{i}})) \leqslant 1$ this constraint also related to the main reason of solving MASIB which is finding long sequences for error detection, so we suggest that all agents sizes will be equal with maximum difference of 1. Maybe one can get higher g value for MASIB by omitting this constraint. But any variation of MASIB without this constraint is out of the scope for this work.

Evaluation

This section will focus on the experiment process: implementation details, experiment definition, and the results analysis.

Implementation The entire code regarding this work can be forked from: https://github.com/YossiCohen/MA-SIB so anyone that is exited and wants to contribute will be able to do that. The main project is a C# project named MaSib it contains implementation for A* and DFBNB algorithms optimized for maximal reward. Since in this problem we need to hold in the search node the entire snake, actually we don't need a pointer from one search node to it's parent search node. If we also ignore the closed list actually we can free the memory allocated for a search node after expending it. in this implementation it is done by the garbage collector since after we finish expanding a node no object will refer to it, this implementation saves great amount of memory when running A*.

The above project holds 3 implementation of domains:

- *Snake* - single agent snake in the box implementation.
- *BoxCartesian* - Multi agent snake in the box implementation with Cartesian product for all agents' moves' when expanding a node.
- *BoxOD* - Multi agent snake in the box implementation with Operator Decomposition.

All above classes implements the *INode* interface therefore can run with A* and DFBNB, notice that both *Box* (MASIB implementations will be called box) holds a collection of snakes so we can use each snake heuristic when growing snakes. There are 3 heuristics for snakes $none$, $legal$ and $reachable$ for box we've offered 3 heuristics (ignoring the $none$):

- $SnakesSum$ This heuristic sums up all snakes heuristics and calculate the number of untouched locations in Qn and returns the minimal value between them two.
- $Legal$ Similar to the legal heuristic of snakes but take into consideration the minimal value between l and k and counts all the legal vertices according to this value.
- $Reachable$ Similar to the reachable heuristic of snakes this heuristic also take into consideration the minimal value between l and k and counts the reachable vertices according to this value when searching from all snakes heads.

Notice that since $SnakesSum$ heuristic sums up all snakes heuristics you need to give each snake a heuristic for it to be valid. On the other hand, if you are using box $legal$ or $reachable$ it is better to set the snakes heuristic to $none$ to save their calculation time.

The solver gets as an input the problem type, number of dimensions, spreads (inter & intra), heuristics (for each snake & for the entire *Box*) and starting locations. It will generate an output log while running. The project have 2 more helper tools, the first is *ExperimentSummarizer* - a small executable (C# code on GitHub - part of this solution) that can be launched in the logs folder, it will generate CSV file that sums up all logs and can be analyzed in tools such as Excel. The second helper tool is python scripting to launch multiple experiments - some examples exists in *Lab* folder in the repository. Since the solver gets as an input the starting locations of the snakes if we would like to solve MASIB for specific number of agents we need to solve the problem for all possible starting locations this is can be done very easily via scripting. All experiments will be done by scripting the desired problems, running the scripts and lastly running the *Experiment Summarizer* to manually analyze the results

[Box-H, Snake-H] \ [Algorithm, Box-Type]	[A*, Box-Cartesian]	[A*, Box-OD]	[DFBnB, Box-Cartesian]	[DFBnB, Box-OD]
[Box-Legal, Snake-None]	172426	24	89395	**23**
[Box-None, Snake-None]	87375	31	86933	26
[Box-Reachable, Snake-None]	202094	141	116796	120
[Box-SnakeSum, Snake-Legal]	313724	171	98434	111
[Box-SnakeSum, Snake-Reachable]	226441	317	100228	119

Table 2: Algorithms, Node generation methods and heuristics comparison, results of average runtime in milliseconds. We've omitted some combinations such as [Box-Reachable, Snake-Reachable] because the box heuristic not taking into consideration the snake heuristic

n \ [**l**, k]	[**2**, 2]	[**3**, 2]	[**3**, 3]	[**4**, 2]	[**4**, 3]	[**4**, 4]	[**5**, 3]	[**5**, 4]	[**5**, 5]	[**6**, 4]	[**6**, 5]	[**6**, 6]	[**7**, 5]	[**7**, 6]	[**7**, 7]
4	6														
5	12	6	6												
6	26	12	10	6	6	6									
7			18		10	10	6	6	6						
8						16		10	10	6	6	6			
9									14		10	10	6	6	6

Table 3: MASIB Maximum length when $|a| = 2$ for the above n,l,k notice that l value appears in **bold**

n \ [**l**, k]	[**2**, 2]	[**3**, 2]	[**3**, 3]	[**4**, 3]	[**4**, 4]
5	11				
6	22	9	9		
7			16	3	3
8			31	13	13

Table 4: MASIB Maximum length when $|a| = 3$ for the above n,l,k notice that l value appears in **bold**

Experiments and Results Analysis In order to decide in what preferences we should run the experiments the first experiment was to run all variations of algorithms and heuristics and decide what formation gives us lowest runtime, more details regarding this experiment in the subsection *performance*, later on we launched another 2 experiments to find the optimal solution for some (n, a, k, l)MASIB on different dimension values, two and three agents, and for every k, l we set number of different spreads values. the guide line in all experiments was that every launch will take maximum time of 15 minutes because of the project's schedule and the fairly large amount of experiments. we've omitted from this paper results that didn't finished on time.

Performance Before running the solver to find the optimal solution for various MASIB problems we wanted to decide what algorithm and heuristics to use when searching. To do that we have run the solver on different MASIB setups. every setup launched with algorithm:[A*, DFBNB], box implementations:[BoxCartesian, BoxOD], snake heuristic:[None, Legal, Reachable] and *box* heuristic: [None, SnakesSum, Reachable, Legal]. the following lists shows the dimensions checked and the spreads checked for every dimension:

- n=5 : spreads[2,3]
- n=6 : spreads[2,3,4]
- n=7 : spreads[3,4,5]
- n=8 : spreads[4,5,6]
- n=9 : spreads[5,6,7]

We have assigned all the combinations on the above spread in k and l (intra snake spread and inter snake spread) where $k \leqslant l$.Running above experiment can be done form:*Lab/masib-runner-performance.py*

In Table 2 you can see the comparison of different combinations of snake and box heuristics and the average runtime on every algorithm. Firstly we can see that Operator Decomposition performs much better than Cartesian product The differences are so enormous that we think we should check for possible improvements in the current implementation. It's also easy to see that box legal heuristic performs well comparing to all other heuristics. On the contrary it can be because of the relatively small problems. since some of the times no heuristic at all perform better than a simple heuristic a good explanations can be that the problems are too small and the heuristic's overhead influence too much on the results.

Optimal MASIB Solutions Table 3 shows the maximal solution for two agents on variety of k and l values for every n dimension. Most of the table is empty since some of data is not very interesting such as big spread values on small dimensions (upper and right areas of the table), on the other hand, some of the data is very interesting but also very hard to get (lower and left areas of the table) - for instance, solving (7,2,2,2)MASIB 127 times for different starting positions of the second snake can take a lot of time. The reader

can review the results while comparing them to the results of single snake from Table 1 an interesting fact we can see is that not only solving the problem as multi agent is better than splitting a single snake; It is sometimes have the same g value, i.e: $(6,2)$SIB have $g = 26$ and $(6,2,2,2)$MASIB have $g = 26$ too. You can examine Table 4 to see more optimal solutions for three agents, this table is relatively sparse because of the multiple runs needed in order to solve a problem, in example, to solve the $(8,3,3,3)$MASIB we need to run the solver approximately $\binom{256}{2} = 32640$ times.

both above experiment can be found at: *Lab/masib-runner-two-snakes.py* and *Lab/masib-runner-three-snakes.py* in the GitHub repository. Another luncher exists in the *Lab* folder, it is named: *masib-solver-optimally.py*. As mentioned above, in order to solve the (n,a,k,l)MASIB we need to try all the starting positions for all $|a|-1$ snakes (assuming s_1 starting location is always 0) this file allows the user to set all the relevant parameters to the solver, such as algorithm, box type, heuristics, spreads and runtime limitation and set the desired number of agents value $|a|$. the script will run the solver $\binom{2^n}{|a|-1}$ times with all the needed starting locations. Later on the user will be able to investigate the logs via the *Experiment Summarizer* tool.

free spots and optimal solutions On SIB problems we can find free spots on optimal solutions. In example, for $(6,2)$SIB example of optimal solution can be: $0-1-3-7-15-13-12-28-20-21-53-37-36-38-46-62-63-59-43-41-40-56-48-50-18-26-10$ this solution have *free spot* located at 25. Does MASIB solutions are free of *free spots*? after reviewing the data our conclusion is: No. Consider the following optimal solution for $(5,2,2,2)$MASIB: $a = [s_1 = (00000, 00001, 00011, 00111, 00110, 01110, 01010), s_2 = (11111, 11011, 11001, 11000, 11100, 10100, 10101)]$ with $g(a) = 12$, this solution have *free spots* on $[01101][10010]$ Ultimately, we can't say that SIB/MASIB optimal solution must be free of *free spots*. When running the solver it will output all the *free spots* of the sequence. If the reader have sequence and want to check it in the aspect of *free spots* it can be done via checking the sequence at: https://yossicohen.github.io/MA-SIB/ it is an on-line *free spots* checker for existing sequences.

Experiments Conclusions After running the experiments and analyzing the results we have come to the following conclusions:

The fact that we've checked only relatively small problems can explain why we've seen that solving the problem without using heuristic is faster than with most heuristics - since the problems' were relatively small the heuristic calculation overhead was greater than it affect. More experiment should be done on bigger and more complicated problems in order to see heuristics affect.

We have shown that solving MASIB as a multi agent problem can produce sequences that are longer than any other method - such as splitting the hypercube or splitting SIB solution. Actually we have examples that shows that $(6,2)$SIB and $(6,2,2,2)$MASIB have the same g value. but is it the limit? Can we get a higher g value on $(n,2,k,k)$MASIB when comparing to matching (n,k)SIB or maybe the (n,k)SIB g value upper bounds $(n,2,k,k)$MASIB g? further experiments must be done to check this empirically - or theoretical proof is needed.

Discussion and Future Work

We have suggested a new problem in this paper - (n,a,k,l)MASIB and discussed how it can be solved via heuristic search methods. We have demonstrated that solving this problem as a multi agent can produce better results than splitting single snakes or splitting the dimensions into smaller hypercubes. On the implementation level, more heuristics can be added to the solution both inter and intra snakes heuristics, moreover, implementation of this problem in a non managed environment can have great affect on the performance.

More experiments can be made with different setups to check if heuristics function better when the problems gets more complicated and to find more optimal solutions for the problem.

Further more work can be done on the aspect for finding the best locations to grow the snakes from, on this work we've used the brute force approach but can it be done in a more sophisticate way? Another aspect that can be researched more deeply is the *free spots* and how can we use them when searching for the best locations to grow the snakes from. Assuming we have a candidate solution with x free spots, and we are searching for other starting location for snakes, if we find that there are more than $f(x)$ unreachable locations on Qn can we prune the node? does this pruning method relevant and efficient in this domain?

Finally we hope this problem will encourage more research for better solution and inspire new ideas in the practical field of error correction.

References

[Kautz 1958] Kautz, W. H. 1958. Unit-distance error-checking codes. *IRE Trans. Electronic Computers* 7(2):179–180.

[Palombo et al. 2015] Palombo, A.; Stern, R.; Puzis, R.; Felner, A.; Kiesel, S.; and Ruml, W. 2015. Solving the snake in the box problem with heuristic search: First results. In Lelis, L., and Stern, R., eds., *Proceedings of the Eighth Annual Symposium on Combinatorial Search, SOCS 2015, 11-13 June 2015, Ein Gedi, the Dead Sea, Israel.*, 96–104. AAAI Press.

[Standley 2010] Standley, T. S. 2010. Finding optimal solutions to cooperative pathfinding problems. In Fox, M., and Poole, D., eds., *AAAI*. AAAI Press.

[Wang and Botea 2008] Wang, K.-H. C., and Botea, A. 2008. Fast and Memory-Efficient Multi-Agent Pathfinding. In *Proceedings of the International Conference on Automated Planning and Scheduling ICAPS-08*, 380–387.