Steiner Triples Cover Experiments and Sparse Ruler Self-Avoiding Multiwalk Stochastic Search

Franc Brglez
Department of Computer Science, NC State University, Raleigh, NC, USA
https://people.engr.ncsu.edu/brglez/
brglez@ncsu.edu

Abstract

Minimum set covering problems arise in a number of domains and contexts. In logistics, the context includes market analysis, crew scheduling, emergency services, etc. In electronic design automation, the context includes logic minimization, technology mapping, and FSM optimization. In bioinformatics, combining Chromatin ImmunoPrecipitation (ChIP) with DNA sequencing to identify the binding sites of DNA-associated proteins can also be formulated as the motif selection problem, mapped to a variant of the set cover problem. Traditionally, these problems have been solved with branch-and-bound solvers which will guarantee an optimum solution — provided they do not time out. However, for large problem instances, branch-and-bound solvers cannot compete with stochastic solvers. This paper introduces a new algorithmic framework based on the concept of sparse ruler self-avoiding multiwalk stochastic search, SPARSS. The uncensored mean first-passage-time experiments, performed with instances of Steiner triples of increasing size, demonstrate that the asymptotic complexity of solvers based on the concept of SPARSS may well outperform the current generation of solvers: both in terms of solution quality as well as runtime.

1 Introduction

Stochastic optimization is the top layer of the 'Big data analytics' pyramid [10]. In contrast to the abstract from a 2013 journal article ‘Metaheuristics – the Metaphor Exposed’ [17]:

“... a true tsunami of ‘novel’ metaheuristic methods, most of them based on a metaphor of some natural or man-made process. The behavior of virtually any species of insects, the flow of water, musicians playing together – it seems that no idea is too far-fetched to serve as inspiration to launch yet another metaheuristic. ...”

this article may read as an antidote. We argue for alternative views in the following sections, including the Appendix section. While these sections are listed in its natural order, we ask the reader to review the Appendix section first. The Appendix section provides the context for the motivation and the terminology we use. Most of our terminology is rooted in traditional definitions from calculus, some may be at variance with the terminology used currently to describe the prevalent metaheuristic methods and metaphors.

Section 2: Set Cover Instance school_5_6

introduces the set cover problem with small set cover problem instance. A school interviews $L = 5$ teachers and finds a single minimum cover solution: three teachers are hired to teach all of the 6 subjects.

The sparse graph in Figure 1a with $2^5 = 32$ vertices represents not only the set of all binary-encoded coordinates to be considered in solving this problem but also the rank of each binary

1 ... corresponding author

© Franc Brglez
Licensed under Creative Commons License CC-BY
Leibniz International Proceedings in Informatics
LIPIcs
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
coordinate $x_{i,r}$ and the value of the objective function $OF(x_{i,r})$. Figures 1b and 1c contrast the simple $O(L = 5)$ neighborhood evaluation for coordinate 10011 versus the tableau $O(2)$ neighborhood evaluation for coordinate 10011.

Section 3: The SPARSS Algorithmic Framework

This section outlines the guiding principles of sparse ruler self-avoiding multiwalk stochastic search, SPARSS, by way of instance school_7_8. Now the school interviews $L = 7$ teachers and finds a single minimum cover solution: three teachers are hired to teach all of the 8 subjects.

The sparse graph in Figure 2 has $2^7 = 128$ vertices. The single minimum cover solution begins to appear as the needle in the haystack problem.

See Figure 3 for concepts of base vertices and coordinates, feasible vertices and coordinates, blocked vertices and coordinates, and finally, multiwalk expeditions launched with $k$ bots from feasible base coordinates in the direction from rank $= r$ to feasible vertices at rank $= r - 1$.

See Figure 4 for a specific instance of a multiwalk expedition that reaches the unique coordinate with valueTarget at rank $= 3$ during the second expedition. Each expedition always starts with $k$ bots from the base of feasible coordinates at rank $= 6$.

See Figure 5 for a near-generic implementation of the SPARSS solver: it lists the complete R-code, executable with a simple command line in R-shell script. What is specific to the set cover problem in this code is the decision to initiate the multiwalk with $k$ bots from one coordinate base only, the base on the left-hand side of the polar graph with the coordinate rank $r_{\text{baseLeft}}$, where $L/2 < r_{\text{baseLeft}} < L$. A generic implementation of the SPARSS supports multiwalks with $2k$ bots starting from two bases, one as described already, the other at rank $r_{\text{baseRight}}$, where $0 < r_{\text{baseRight}} < L/2$. Given sufficient computational resources, this code will find the best of near-best solution efficiently to any problem formulated by a set cover instance.

Section 4: The First Experiments with Steiner Triples

Instances of Steiner triples coverings provide an excellent set of test cases that have challenged set cover solvers for a number of years. This challenge is still on-going [15]. As shown in Figure 6, an asymptotic progression of first-passage time experiments with our solver offer new insights.

Section 5: Summary and Conclusions

We have proposed and prototyped in R a new algorithmic framework based on the concept of sparse ruler self-avoiding multiwalk stochastic search, SPARSS. The framework is generic and can be readily adapted and adopted for stochastic search in a number of domains formulated with permutation coordinates, $b$-ary coordinates, and composites of such coordinates. The multiwalk concepts extends the capabilities of the single self-avoiding walks that already outperformed a number of state-of-the-art solvers to date.

Appendix A: Graph Cover Time and Complete Sparse Rulers

Concepts of sparse rulers and graph cover time under multiple walks continue to motivate and guide our approach to stochastic optimization. For an example of sparse ruler, see Figure 7a. For an example of a 5-regular sparse graph, see Figure 7b. Under the notation of $[0,1]^L$, mathematicians will say that this graph represents a 'flattened' hypercube, associating vertices on the cube with binary strings of length $L$. In our context, we recognize the graph as a layered polar graph, with each layer defined by the rank of the binary coordinate, ranging from 0 to $L$.

We define the left-most vertex as having the rank $= 0$ and the right-most vertex as having the rank $= L$. For binary coordinates, the rank is simply defined by the number of 1’s in the string.

Our key principle in making stochastic search more efficient relies on reducing the number or repeated visits to same vertices, taking place during unbiased random walks. To date, our most successful stochastic search strategy to reduce the number of repeated visits uses the concept of the self-avoiding walk. The generalization to multiwalks leads to furcated self-avoiding walks, as illustrated in Figure 7b. The only coordinates that walking bots are allowed to visit repeatedly by definition, are the feasible base coordinates, predetermined before the start of any multiwalk expedition.

Minimum set covering problems arise in a number of domains and contexts. In logistics, the context includes transportation, market analysis, crew scheduling, emergency services, etc. [1].
In electronic design automation, the context includes logic minimization, technology mapping, and FSM optimization [12, 13]. In bioinformatics, combining Chromatin ImmunoPrecipitation (ChIP) with DNA sequencing to identify the binding sites of DNA-associated proteins can also be formulated as the motif selection problem, mapped to a variant of the set cover problem [14]. Instances of Steiner triples coverings provide an asymptotic progression that continues to challenge set cover solvers. For a comprehensive article on Steiner triples coverings, including the best-known-solutions for two new largest instances, see [15].

2 Set Cover Instance school_5_6

This section introduces the set cover problem with a small set cover problem instance. A school interviews $L = 5$ teachers and finds a single minimum cover solution: three teachers are hired to teach all of the 6 subjects.

The sparse graph in Figure 1a with $2^5 = 32$ vertices represents not only the set of all binary-encoded coordinates to be considered in solving this problem but also the rank of each binary coordinate $x_{i,r}$ and the value of the objective function $OF(x_{i,r})$. Figures 1b and 1c contrast the simple $O(L = 5)$ neighborhood evaluation for coordinate 10011 versus the tableau $O(2)$ neighborhood evaluation for coordinate 10011.

Instance description and Objective Function evaluation.

The set cover is described in cnf format that is often associated with instances of the sat problem. This format is convenient since in many scenario the unate clauses that define the set cover can be readily interpreted as binate clauses when some of the integer variables in the clause are negative. The need to find minimum binate covers arises when solving problems in Electronic Design Automation (EDA). In order to find the minimum binate cover, we need to solve the sat problem first [13]!

The R-function that returns the value of the objective function requires counting the number of clauses not covered by a given binary coordinate. Examples in the next two paragraphs illustrate the counting process. We say that a coordinate that covers ‘all clauses’ is feasible. For each feasible coordinate, the value returned by the objective function is simply the the rank of its coordinate:

$$valueOF = rank(coord)$$

A coordinate that does not covers ‘all clauses’ is unfeasible. The value returned by the objective function includes rank and a penalty function with the count notCovCnt and $L$:

$$valueOF = rank(coord) + notCovCnt + L$$

Evaluation of the coordinate neighborhood: the simple method is $O(L)$.

Figure 1b illustrates a simple method evaluation of the of pivot coordinate at rank = 3 and all of its $L = 5$ adjacent coordinates at ranks 2 and 4. Here, all evaluations are independent of each other. We argue that the complexity of evaluating all neighbors using the simple method is on the order of $O(L)$.

Evaluation of the coordinate neighborhood: the tableau method is $O(2)$.

Figure 1c illustrates a tableau method evaluation of the of pivot coordinate at rank = 3 and all of its $L = 5$ adjacent coordinates at ranks 2 and 4. Here, evaluations depend (1), on nominal evaluation of the pivot coordinate, and (2), on $L$-column evaluations that represent coordinate changes only with respect to the pivot coordinate. The first evaluation stores
Figure 1 An introduction of a set cover problem instance: (a) school interviews 5 teachers and finds a single minimum cover solution; (b) simple $O(L = 5)$ neighborhood evaluation for coordinate 10011; (c) tableau $O(2)$ neighborhood evaluation for coordinate 10011.

The binary vector isCov that marks whether the clause is is covered or not. The $L$-column evaluations consist of flipping each of $L$ coordinate bits and counting the numbers that flipped in vector isCov with respect to the first evaluation. It is apparent that this method of evaluating $L$ neighbors of the pivot coordinate is considerably more efficient than the simple method. We argue that the complexity of evaluating all neighbors using the tableau method is on the order of $O(2)$.

Neighborhood transparency under integer and binary encoded coordinates.

Consider the line plot that depicts $OF(index)$ versus index in Figure 1a. $OF(index)$ reaches the minimum value of 3 at index = 19. Under the natural order of index as the integer coordinate we may think there are only two neighbors from which we can reach the minimum value at index = 19: indices 18 and 20. However, observing the neighborhood relations under the binary coordinates, the minimum value can be reached from five adjacent coordinates. Converted to integers, the indices are 3, 17, 18, 23 and 27. The line plot that suggests index
Figure 2 A line graph induced by the set cover instance school_7_8. The graph displays values of objective function $OF(index)$ versus index that represents $2^7 = 128$ binary coordinates. There are a total of 22 feasible coordinates that return $OF$ value $\leq 7$ with a single minimum value of 3 for index = 69 (a binary coordinate 1000101). Given this information, this instance begins to appear as the needle in haystack problem.

20 is a ‘neighbor’ from which we can reach a minimum value of 3 is therefore wrong and misleading. This problem has been defined to be evaluated in binary coordinate domain. However, the line plot does correctly report 5 feasible solutions!

3 The SPARSS Algorithmic Framework

This section outlines the guiding principles of sparse ruler self-avoiding multiwalk stochastic search, SPARSS, by way of a larger instance of a set cover problem, the instance school_7_8. Now the school interviews $L = 7$ teachers and finds a single minimum cover solution: three teachers are hired to teach all of the 8 subjects. There are $2^7 = 128$ coordinates and a single minimum cover solution begins to appear as the needle in haystack problem, suggested by the line graph in Figure 2.

The graph segment in Figure 3 highlights a number of features that guide the design of the SPARSS algorithmic framework.

- each vertex label is a triplet: rank,index:value prefixed with v. Here, index represents the integer value of the binary encoded string. The label v3,169:3 represents a coordinate 1000101 with the rank of 3 and the set cover value = 3.
- separation of vertices/coordinates into feasible and unfeasible: any coordinate where the coordinate rank equals the value returned by the set cover objective function is denoted as feasible.
we distinguish three types of edges between adjacent vertices:

- **feasible edges** between two feasible vertices
- **unfeasible edges** between two unfeasible vertices
- **blocked edges** between a feasible vertex and an unfeasible vertex

**coordinate base** is a set of coordinates with the same rank. The upper limit on the size of this set is the binomial coefficient. In R, we use the function `choose(L, r):` $L$ denotes the length of the binary string, and $r$ denotes the value of the coordinate rank.

The base coordinates are separated into feasible and unfeasible coordinates. A **multiwalk expedition** is launched with $k$ bots that are assigned from the subset of feasible base coordinates. Afterward, all steps taken by bots are from feasible vertices at rank = $r$ to feasible vertices at rank = $r - 1$ only – until blocked and returning to base if the target value is not found first. The choice of walk direction from higher rank to lower rank is specific to the set cover problem.

Figure 4 illustrates a number of multiwalks with two bots, including both bots getting blocked during the first expedition. During the second expedition, one bot reaches $value_{Target}$ and terminates the multiwalk. Both bots started by randomly selecting coordinates from the base at rank = 6, returning to the base when both were blocked at rank=3. After randomly selecting coordinates from the base at rank = 6 again, one of the bots succeeds to reach the unique coordinate with $value_{Target}$ at rank = 3. For a comprehensive description of all phases of the multiwalk, also supported by a detailed transcript returned by the solver, see the caption of Figure 4. Most intriguing is the observation of walk furcation from a vertex that is not a base vertex where such furcations are expected to occur by the design.
Figure 4 A graph segment induced by the set cover instance school_7_8. Here, at the expedition base with rank = 6, two bots start the multiwalk expedition from two feasible coordinates. After making two steps each, both bots are blocked at rank = 3 and return to the base at rank = 6. Both bots again choose randomly two feasible coordinates at the base and start a new multiwalk expedition. Both complete step 1 at feasible vertices v5,55:5 and v5,87:5. While probing for feasible bots at rank=4, bot at v5,55:5 is blocked at v5,23:12 while bot at v5,87:5 reaches a feasible vertex at v4,71:4. On the next probe, bot at v5,55:5 is blocked again at v4,51:13 and bot at v5,87:5 is blocked at v4,83:12. The algorithm is designed to probe additional new neighborhood coordinates for as long as possible or until it reaches the predetermined quota of finding 2 feasible coordinates if possible. Hence one more probe from v5,87:5, now finding a feasible vertex at rank=4: the vertex v4,85:4. With two feasible coordinates again, v4,71:4 and v4,85:4, the search proceeds towards rank=3. Now, the bot at v4,71:4 is blocked at v3,7:12, and the bot at v4,85:4 finds the target vertex v3,69:3. The search stops ‘uncensored’ after a total of 23 probes and 8+1=9 steps. Most notably, observe that walk furcation of the vertex v5.87:5. This furcation demonstrate the generality and the significant merits of current implementation!

This section concludes with Figure 5 that presents the near-generic implementation of the SPARSS: it lists the complete R-code, executable with a single command line in R-shell script. What is specific to the set cover problem in this code is the decision to initiate the multiwalk with $k$ bots from one coordinate base only, the base on the left-hand side of the polar graph with the coordinate rank $r_{\text{baseLeft}}$, where $L/2 < r_{\text{baseLeft}} < L$. A generic implementation of the SPARSS supports multiwalks with $2k$ bots starting from two bases, one as described already, the other at rank $r_{\text{baseRight}}$, where $0 < r_{\text{baseRight}} < L/2$. Given sufficient computational resources, this code will find the best of near-best solution efficiently to any problem formulated by a set cover instance.

The report in Figure 5 summarizes an experiment using a Steiner instance with 281 coordinates. When $\text{valueTarget}$ was relaxed from optimum of 61 to 63, the instance was solved in less then a 45 seconds on a MacBook. A multiwalk solution with 4 bots required only a single expedition. The bots probed the objective function 337 times during a total of 65 steps! Repeated trials returned equivalent solutions with small runtime variance.
Figure 5 A near-generic implementation of the SPARSS algorithmic framework. This is a verbatim listing of the complete R-code, executable with a single command line in R-shell script. The function `solv_bina_setc_init` is called only once to read the set cover instance and to setup and return the vector of feasible coordinates that establish the base of feasible coordinates at rank = `rankRLBase`. The letters ‘RL’ imply that the base is located near the maximum rank of the polar graph, i.e. on the ‘right’ so that each walk will proceed in the direction of descending rank – a feature expected when we are solving a set cover problem. The inner core of the solver are two nested loops: for (j ...) and for (i ...), computing the multi-neighborhood of adjacent coordinates `cNeighbLM` from which we extract the feasible coordinates `cFeasible` that are needed by bots before proceeding to next steps of the multiwalk. The critical feature is the use of the `hash()` utility: satisfying the condition if (!hash.key, cAdj, cUrn) ensures that we have no replicated coordinates when evaluating the objective function.

4 The First Experiments with Steiner Triples

Instances of Steiner triples coverings provide an excellent set of test cases that have challenged set cover solvers for a number of years. This challenge is still on-going. A comprehensive article that includes best-known-solutions for two new largest Steiner triples instances to date is most likely this one [15].

The best known solutions for these instances, as reported in [15], are tabulated for context in Figure 6. A line graph displaying solutions for the smallest instance of Steiner triples, with 9 variables and 12 clauses is included for comparison with the much smaller 7-variable, 8-clause instance `school_7_8`. We use this instance to test features of our solver. As shown in in Figure 6, the smaller instance `school_7_8` has a single minimum solution, compared to 54 solutions found for the Steiner triple instance. The relatively large number of available solutions is clearly a factor why this instance is being solved with our solver in 2 to 3 probes of the objective function.
Figure 6 The initial test results with the SPARSS solver running R-code introduced in Figure 5. This code uses the simple method to probe the neighborhoods during the multiwalk. The implementation of the much more efficient tableau method outlined in Figure 1 is work in progress and will be reported in the next version of this draft. The listing of best-known-solutions for Steiner triple covering is reproduced from [15].

The asymptotic progression of first-passage time experiments tabulated for our solver are just tests for correctness. Statistically significant first-passage time experiments will be reported in the next draft after the completion of the tableau method implementation.

After the completion of our R-prototype I invite a researcher with experience in C/C++ programming to collaborate in advancing this research to large set cover instances. The uncensored mean first-passage-time experiments may well be the best method to compare the SPARSS solver with the current generation of set cover solvers.

5 Summary and Conclusions

We have proposed and prototyped in R a new algorithmic framework based on the concept of sparse ruler self-avoiding multiwalk stochastic search, SPARSS. The framework is generic and can be readily adapted and adopted for stochastic search in a number of domains that are formulated with permutation and $b$-ary coordinates. The multiwalk concept extends the capability of the single self-avoiding walk that has already outperformed a number of state-of-the-art solvers to date.

The set cover formulation in this paper is just a vehicle to prototype and validate the concepts we introduced. As ‘discovered’ in the concluding caption of Figure 4, the nature of multiwalk as we define it is highly dynamic; the ‘furcation’ of walks we observed for this particular initial seed may not be the last ones we observed, more surprises and explanations are anticipated.
Most of the terminology we use is rooted in traditional definitions from calculus and some may be at variance with the terminology used currently to describe the prevalent metaheuristic methods and metaphors. The best way make a case for any stochastic search method is a series of well-designed set of controlled and asymptotic experiments that measure and report, with sufficient precision, the uncensored mean first-passage-time [16]. The notion of first-passage-time is a metaphor not only for the platform-specific runtime but more importantly, for the platform-independent number of objective function evaluations.

A Graph Cover Time and Complete Sparse Rulers

Random walks and multiple random walks have been studied on graphs in the context of interaction between particles [9], [8]. The cover time of a random walk on a random $r$-regular graph was studied in [9] where it was shown with high probability (whp) that for $r \geq 3$ the cover time is asymptotic to $\Theta(n \log(n))$, where $\Theta_r = (r - 1)/(r - 2)$. The follow-up article [8] shows that for $k$ independent walks on a regular graph $G$, the cover time $C_G(k)$ is asymptotic to $C_G/k$, where $C_G$ is the cover time of a single walk.

Representations of integers $1, 2, \ldots, N$ by differences have been studied by a number of mathematicians [11]. More recently, this effort continues under the label of sparse rulers [18].

Both concepts, multiple walks and sparse rulers, have motivated our approach to stochastic
optimization, starting with a short invited article in 2018 [5]. This appendix illustrates the most relevant definitions that support the notation used in the body of the current article on the set cover problem. A more comprehensive treatment on these concepts is work in progress:

Natural Order, Symmetry, Sparse Rulers, Cover Time, and Multivalks: the Third Principles in Experimental Stochastic Optimization. Sections will include complete sparse rulers, cover time in regular graphs and neighborhoods under (a) real axis coordinates, (b) binary coordinates, (c) permutation coordinates as well as sections on representative problem instances in each of the coordinate domains.

Look for it end of February 2020: https://people.engr.ncsu.edu/brglez/publications.html

Complete Sparse Rulers. An example of a complete sparse ruler is shown in Figure 7a. The ruler is defined on the range [0,31]. Our representation of the ruler as a bipartite graph is unusual and is designed to induce an association with graph cover time in the 5-regular graph in Figure 7b.

The number of vertices in this bipartite graphs is $N = 2^2 = 32$. For example, the rank of coordinate $u$ in the graph of which $u$ is a member.

(1) Cover time $C_u$ is the expected number of steps for a walk starting at $u$ to reach every vertex in the graph of which $u$ is a member.

(2) The cover time of a graph $G$ is the maximum cover time $C_u$ when considering all vertices $u$ in $G$.

Under the notation of $[0,1]^L$, mathematicians will say that this graph represents a “flattened” hypercube, associating vertices with binary strings of length $L$. In our context, we recognize the graph as a layered polar graph with each layer defined by the rank of the binary coordinate where the range ranges from 0 to $L$. We define the left-most vertex as having the rank = 0 and the right-most vertex as having the rank = $L$. For binary coordinates, the rank is simply defined by the number of 1’s in the string.

In our work, the notion and definition of rank is universal for coordinates defined by strings of numbers, not only binary strings. In [4] we defined rank on composite coordinates represented as a concatenation of binary and ternary coordinates. In [7], we defined rank on permutation coordinates. Current work in progress relates to the definition of rank on quaternary coordinates. The distribution of binary coordinates sorted by rank is binomial. The distribution of permutation coordinates sorted by rank is Mahonian. However, while the polar graphs with $2^L$ vertices are sparse and $L$-regular and graphs with $(L!)$ vertices are sparse and $(L - 1)$-regular, the graphs with $2^L$ and $4^L$ vertices are not regular. For example a graph with $2^4$ vertices has vertices with degrees ranging from 4 to 8 while its rank is ranging from 0 to 8. However, all graphs considered in our work on stochastic optimization are layered and sparse polar graphs with well-defined rank, whether they have have $L!$, $2^L$, $(2^L)(3^L - 1)$, $(2^L)(L!)$, or $(b^L)$ vertices, where $L$ is the length of the coordinate string and $b$ is the coordinate base under $b$-ary encoding. It is also interesting that layered and sparse polar graphs graphs with $(2^L)(L!)$ vertices are $(2L - 1)$-regular graphs with maximum rank of $L(L - 1)$. Each coordinate represents a signed permutation string. For example, the rank of coordinate $-2, +3, +1$ is 3 and two of its adjacent coordinates $+2, +3, +1, -2, -3, +1$ have ranks of 2 and 4. Our early work in [3] associates such graphs with $L$-dimensional dice and Hasse graphs since these graphs are direct descendants of the well-known Hasse diagrams. In [3], Hamiltonian paths are illustrated for graphs with $2^L$ and $(2^L)(L!)$ vertices. Of course, for large values of $L$ such graphs cannot read and stored in computer memory. However, our experiments demonstrate that the well-defined coordinate rank and the neighborhood structure of these graphs remains and will remain the fundamental data structure to advance global optimization and stochastic search methods today and in the future.

Global Optimization under Stochastic Search. Consider $N$ vertices in the two graphs as integer coordinates $x_I$ in Figure 7a and as binary coordinates $x_B$ in Figure 7b. Let $\Theta_I(x_I)$ and $\Theta_B(x_B)$ denote two functions defined on these coordinate: our objective is to devise an efficient method to find a global minimum for each function. In the worst case, we would need to visit all
Figure 8 An illustration of a layered graph section with vertex labels represented with binary encoded strings of length $L = 7$ where ranks of each string range from 0 to $\lfloor L/2 \rfloor = 3$. Vertices at each rank can serve as a coordinate base for $k \geq 1$ bots to start the self-avoiding multiwalk. In this example, steps taken by each bot are directed to successive coordinate layers of increasing rank. Also, each pair of adjacent layers can also be viewed as a special case of a complete sparse ruler.

$N$ vertices before finding the minimum, so that even the unbiased multiple random walk such as formulated in [8] is more expensive on the average than the exhaustive search.

Our key principle in making stochastic search more efficient is based on reducing the number or repeated visits to the same vertex that takes place during any unbiased random walk. Each of the graphs we formulated in this appendix has a Hamiltonian path, i.e. a Hamiltonian walk visits every vertex once with no repeats. However, such walks provide no advantage over exhaustive search.

By now, our most successful stochastic search strategy relies on the concept of the self-avoiding walk [4, 7, 2, 6, 5]. In each case, our solvers outperform the alternative solvers. In this article we propose and implement further improvement by implementing self-avoiding multiwalks. As suggested in Figure 7b, all vertices in the graph are covered by two trees, with one root vertex at rank = 0 on the left, and one root vertex at rank = $L$ on the right. Assuming that the vertex with the minimum value will be found at one of the coordinates with rank = 2 or 3, we can designate all vertices at rank = 1 as base[1] and all vertices at rank = 4 as base[4]. We can then assign 4 bots to start walking: 2 bots on the left will be randomly assigned two coordinates at base[1] and will make the first 2 random steps to the right while 2 bots on the right will be randomly assigned two coordinates at base[4] and make the first 2 random steps to the left. If the target value (minimum) is not found by either pair, we will say that both pairs of bots were blocked and coordinates at rank = 2 and rank= 3 will be marked as visited. Then both pairs of bots will return to randomly selected coordinates at their respective bases. The next steps for both bot pairs will be random again, the bot that encounters a coordinate marked as ‘visited’, will return to base and wait. As soon as one of the bots reaches the target value, a flag is raised that the minimum has been found and the search terminates. A generic implementation of these concepts is illustrated with an executable and literally verbatim R-code in Figure 5 of this paper. Given sufficient computational resources, this code will find a solution efficiently to any problem formulated by a set cover instance. Notably for set cover instances, the base coordinates are defined at rank near the maximum rank of $L$ only. For the set cover problem, there is little if anything to be gained by introducing another coordinate base at rank near the minimum rank of 0.

The multiwalks in Figure 7b are but a small example of furcated self-avoiding walks since the only coordinates that bots are allowed to visit repeatedly are the predetermined coordinates at base[1] and base[4]. For an example of a larger layered graph segment and a larger tree of potential coordinate bases, see Figure 8 and the remainder of this paper.
B References


