Exact and Approximate USCP With Branch and Bound

Janez Radeščček
Matjaž Depolli*
janez.radescek@gmail.com
matjaz.depolli@ijs.si
Jožef Stefan Institute
Ljubljana, Slovenia

ABSTRACT

We propose a parallel solver for the unicost set covering problem based on branch and bound approach. The main contributions of this algorithm lay in its fast parallel execution and the ability to work in approximate mode. We demonstrate the proposed approach on the problem instances from GECCO 2021 unicost set covering competition. To tackle the presented problem instances of varying difficulty, we use an automatic tuning of the algorithm's parameters. The results show all the instances can be solved but the performance remains weak on large instances.

CCS CONCEPTS

• **Computing methodologies** → **Shared memory algorithms**; **Combinatorial algorithms**; • **Mathematics of computing** → *Trees*.

KEYWORDS

USCP, set covering, branch and bound, parallel

ACM Reference Format:
Janez Radeščček and Matjaž Depolli. 2021. Exact and Approximate USCP With Branch and Bound. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3449726.3463298

1 MOTIVATION AND APPROACH

The motivation for the experiment presented here stems in an effort to develop a general framework for implementing parallel branch and bound algorithms in C++. The framework originates from the max-clique algorithm [4], and through the implementation of Unicost Set Cover Problem (USCP), its generality should improve. Furthermore, we are interested in expanding the framework to also cover approximate solvers.

While a time limit to stop the algorithm mid-work is a valid approach for converting the exact branch and bound algorithm into approximate one, this approach often results in very poorly explored search space and the returned solution may be arbitrarily far from optimum. Therefore our efforts were focused on enhancing exploration on the account of decreased local optimization.

Several enhancements to exploration were designed, yet after the preliminary tests, only two most effective were implemented to support the use of time limit. These two were used along with a heuristic for setting their parameters and a method for enforcing memory usage limit, to create an algorithm with a single parameter (time limit) that is able to solve a broad range of problems.

2 UNICOST SET COVERING PROBLEM

USCP is a well known NP-complete problem with a long history [5]. While exact solutions can be obtained for small problem instance in general, in real-life problems, often only an approximate solutions will be obtainable within a reasonable time limit.

Given a finite set X of size n and a family $F = \{S_i \subseteq X\}$ of subsets of X, also called candidates, USCP requires the smallest family $F_{min} \subseteq F$ be found, such that each element from X belongs to at least one set from F_{min}. In other words, for the set X and its cover F, find the smallest subcover F_{min}.

3 EXACT ALGORITHM

The exact algorithm was developed anew, based on a branch and bound principle, similar to existing algorithms [3]. Additional care was taken to support parallel execution through multi-threaded, adapting the method from [4]. A branch is represented with states of elements (covered or uncovered) and states of candidate sets (used, unused or discarded). The algorithm starts with an empty cover. In each call it sorts the remaining candidates in descending order by the number of remaining elements that they cover, and then selects the first one for further exploration, which is a method also used in the greedy algorithm [2]. Unlike with the greedy algorithm, the exact algorithm uses the selected candidate set to split a branch in two and then explores both. The selected candidate set is marked as used on one branch and as discarded on the other. The best solution found so far is cached to bound the exploration - branches with their lower bound higher than the number of used candidate sets in the best solution are pruned.

The lower bound is the number of candidate sets already used in addition to the number of candidate sets required to cover the remaining uncovered elements under the assumption that all the remaining candidate sets are disjoint on uncovered elements. Such a conservative approach to calculating lower bounds ensures the execution will not be pruned and can therefore be found, given enough time. The associated price is a large search space to explore, which makes it unrealistic for the algorithm to finish execution for all but very small problem instances.

4 APPROXIMATE ALGORITHM

In order to try to solve the run-time and memory consumption problems we use three approaches to approximation.

The first considered approach is a limit in the number of sub-branches allowed to be explored on each level of the search tree. Only the first level of the search tree is explored in full, that is, the first candidate set to be used in the cover, while an absolute limit x is applied on all further levels. That means that only the best x candidates, as selected by the greedy heuristic, will be tried on each step of the incremental cover expansion.

The second approach is to reduce the problem size, solve the reduced problem in either the exact or approximate mode - depending on the problem size, and then extend the solution to the original problem. We designed an algorithm in which the elements are iteratively split into multiple disjoint sets to form reduced sub-problems, while the union of their solutions represents the solution to the whole problem. Each disjoint set of elements is created by a heuristic; the uncovered elements are decimated until the fraction r of them remain, in a way that is partly random but also tries to keep the elements covered by a similar subset of candidate sets together. The candidate sets to cover the subproblem are generated as all the candidate sets that at least partly cover the subproblem element set. Finally, the subproblem, which is of significantly lesser complexity than the original problem, is solved. This procedure iterates until the cover of the original problem is complete.

The last approach is to keep a timeout t to interrupt the search if it does not complete in time. Since the order of exploration is greedy, the timeout acts as a means of adjusting between the greedy and exhaustive exploration of space. The same holds even if the already approximate algorithm is interrupted before it finishes, since it also uses greedy approach to exploration. The difficult part in using timeout is in tuning the approximate algorithm parameters in a way that the algorithm will have explored a large portion of its reduced search space when the timeout occurs.

The proposed approximate algorithm combines all approaches. It dynamically selects undefined constants x and r, given t and the size of the problem. It uses functions of $x(t)$ and $r(t)$ that were fitted to some of the preliminary testing results. In addition it uses a memory usage reduction technique, to have the algorithm finish instead of crash on the largest problem instances, although again on the account of opportunistically pruning the search space when additional branches cannot be stored in memory. Since memory requirements are nearly linear with the number of threads, the decision was made to disable parallel execution for problem sizes above an empirically set threshold.

5 RESULTS

The presented algorithm has been implemented in C++ with the use of standard libraries only. It was only tested on USCP part of the problem instances from the *Optimal Camera Placement Problem (OCP) and the Unicost Set Covering Problem (USCP)* [1]. The computer used had dual 4-core 2.3 GHz Intel Xeon E5520 processors and 12 GB of RAM. The preliminary results were obtained with the exact algorithm, limited to 24 hours and to single-threaded execution, and through experiments with varying approximation parameters and time limits are presented in Table 1.

Table 1: Preliminary results

Problem instance	Exec time [s]	Cover exact	best
AC 01	10871	7	7
AC 02	31.5	4	4
AC 03	2.33	3	3
AC 04	DNF	5	5
AC 05	DNF	10	9
AC 06	DNF	15	14
AC 07	DNF	26	24
AC 08	DNF	38	36
AC 09	DNF	55	54
AC 10	DNF	23	23
AC 11	DNF	83	81
AC 12	DNF	173	171
AC 13	DNF	299	297
AC 14	DNF	441	441
AC 15	DNF	641	641
AC 16	DNF	1084	1084
AC 17	DNF	1675	1675
AC 18	DNF	2393	2393
AC 19	DNF	3203	3203
AC 20			4137
AC 21			5291
AC 22			6422
AC 23			7783
AC 24			9231
AC 25			10800
AC 26			12448
AC 27			14353
AC 28			16234
AC 29			18348
AC 30			20555
AC 31			22816
AC 32			25243
RW 01	DNF	867	864
RW 02	DNF	1011	1008

Problem Instance	Exec time [s]	Cover size	best
RW 03	DNF	985	980
RW 04	DNF	1118	1117
RW 05	DNF	1197	1193
RW 06	DNF	1232	1232
RW 07	DNF	1273	1273
RW 08	DNF	1373	1372
RW 09	DNF	1258	1256
RW 10	DNF	1347	1346
RW 11	DNF	421	418
RW 12	DNF	418	417
RW 13	DNF	1709	1706
RW 14	DNF	445	443
RW 15	DNF	456	453
RW 16	DNF	669	668
RW 17	DNF	690	689
RW 18	DNF	441	438
RW 19	DNF	470	468
RW 20	DNF	1873	1869
RW 21	DNF	2069	2069
RW 22	DNF	509	509
RW 23	DNF	546	543
RW 24	DNF	1132	1129
RW 25	DNF	1270	1269
RW 26	DNF	592	590
RW 27	DNF	656	653
RW 28	DNF	843	837
RW 29	DNF	982	979
RW 30	DNF	1421	1420
RW 31	DNF	1618	1615
RW 32	DNF	843	841
RW 33	DNF	960	956
RW 34	DNF	797	795
RW 35	DNF	885	883
RW 36	DNF	807	805

The exact algorithm was limited do single-threaded because its memory requirements were found to be unsustainable because of the large number of branches that have to be stored for backtracking. The size of individual branch is a linear combination of the number of candidate sets and the number of elements to cover, and the maximum number of branches stored per thread is of the magnitude equal or higher than the number of candidate sets in the optimal solution.

The resulting execution time is listed for instances where the algorithm finished by itself, while `DNF` is listed where it timed out. Cover size and execution time are left blank for instances that crashed due to running out of memory. Finally, the best cover size from preliminary experimentation with approximate approaches are listed in the last column. The results demonstrate how broad the spectrum of problem instances is, and the difficulty of solving them with the exact algorithm.

ACKNOWLEDGMENTS

The authors would like to acknowledge the financial support of the Slovenian Research Agency research core funding No. P2-0095 and research funding No. N2-0171.

REFERENCES

[1] Mathieu Brévilliers, Julien Lepagnot, and Lhassane Idoumghar. 2021. *GECCO 2021 Competition on the Optimal Camera Placement Problem (OCP) and the Unicost Set Covering Problem (USCP).* http://www.mage.fst.uha.fr/brevilliers/gecco-2021-ocp-uscp-competition/gecco_2021_ocp_uscp_competition.pdf

[2] V. Chvatal. [n.d.]. A Greedy Heuristic for the Set-Covering Problem. *Mathematics of Operations Research* 4 ([n. d.]), 233–235. https://doi.org/10.1287/moor.4.3.233

[3] Emir Demirovic, Théo Le Calvar, N. Musliu, and K. Inoue. 2016. An Exact Algorithm for Unicost Set Covering.

[4] Matjaž Depolli, Janez Konc, Kati Rozman, Roman Trobec, and Dušanka Janežič. 2013. Exact parallel maximum clique algorithm for general and protein graphs. *Journal of chemical information and modeling* 53, 9 (2013), 2217–2228.

[5] T. Grossman and A. Wool. 1997. Computational experience with approximation algorithms for the set covering problem. *European Journal of Operational Research* 101, 1 (1997), 81–92. https://doi.org/10.1016/s0377-2217(96)00161-0