Hosting of the CG:SHOP Challenges

Phillip Keldenich
Technische Universität Braunschweig, 38106 Braunschweig, Germany

Rouven Kniep
Technische Universität Braunschweig, 38106 Braunschweig, Germany

Dominik Krupke
Technische Universität Braunschweig, 38106 Braunschweig, Germany

Abstract

This paper offers insights into the development and hosting of the *Computational Geometry: Solving Hard Optimization Problems* (CG:SHOP) Challenges. We examine the architecture of the supporting infrastructure, detail the tasks and workloads involved, discuss methods for selecting an effective set of instances, and share the lessons learned from the first seven iterations of the competition.

Keywords and phrases Computational Geometry, Competition, Instance Selection, Optimization

Digital Object Identifier 10.57717/cgt.v5i4.82

Funding This work was partially supported by the German Research Foundation (DFG), project "Computational Geometry: Solving Hard Optimization Problems" (CG:SHOP), grant FE407/21-1.

1 Introduction

Hosting a competition requires considerable effort, but under the right circumstances, it can provide meaningful enrichment to the research community. First, a competition promotes algorithm engineering research on a specific problem, potentially leading to new scientific insights. Competitions offer participants motivation, a structured timeline, and formal recognition for their work. Further, competing as a team fosters collaboration and serves as an effective team-building exercise, indirectly encouraging further scientific advancements. Finally, competitions offer an accessible entry point for students, helping to integrate them into the research community.

Recent CG:SHOP Challenges have drawn substantial participation from student groups, prompting us to focus on enhancing their experience. The extended several-month duration of the competition allows it to cover more of the academic semester, enabling instructors to incorporate participation as part of a lab course. To encourage student involvement, we introduced a junior category so that students do not feel discouraged by competing against more experienced teams. Additionally, the competition problems are designed to be accessible, requiring minimal specialized knowledge in Computational Geometry to get started. Kadıoğlu and Kleynhans [44] further discuss the benefits, experiences, and requirements of using competitions in educational contexts.

The CG:SHOP Challenges began as a workshop during CG Week 2019. The inaugural competition, *CG:SHOP 2019*, ran from February 28 to May 31, 2019, and focused on optimizing the area of a polygon over a fixed set of points. Detailed information on this competition and its outcomes is available in [35]. Due to its success, the competition became an official part of *CG Week*, with top teams invited each year to submit and present abstracts detailing their approaches, which are included in the conference proceedings. In its second iteration, *CG:SHOP 2020* [34], the competition ran from September 30, 2019, to February 14, 2020, and focused on partitioning a point set into convex areas. The following challenges continued to explore diverse problems, including parallel motion planning in *CG:SHOP 2021* [36], intersection-free partitioning of graphs in *CG:SHOP 2022* [37], minimum coverage by convex polygons in *CG:SHOP 2023* [39], and maximum polygon packing in *CG:SHOP 2024* [38]. The seventh challenge, *CG:SHOP 2025*, addressed minimum non-obtuse triangulations.

This paper is structured as follows: We first give an overview of how other competitions are managed. Then, we describe the server architecture and development, including the frameworks and components used, which may serve as a useful guide for those developing their own competition platform. We then discuss the workload associated with organizing a competition, providing an overview of the time and energy required. Afterwards, we explore methods for selecting a high-quality instance set using data science techniques. Finally, we conclude with key lessons learned from organizing these competitions.

2 Other competitions

This section provides an overview of how other competitions are managed, highlighting common practices while noting important differences. The examples were selected based on popularity, similarity, and diversity; the list is not exhaustive. A more extensive, though no longer maintained, collection of competitions can be found at `https://www.hsu-hh.de/logistik/research/challenges`. Since competitions evolve over time, the following overview may not fully represent all editions.

One of the oldest optimization events is the DIMACS Challenge series [5], held since 1990. These events differ substantially from most modern competitions and more closely resemble workshops. Each challenge focuses on a popular problem class in discrete optimization, such as Steiner tree problems or shortest paths, but includes multiple variants and diverse performance metrics. In addition to solution quality, metrics may include measures such as primal integrals [33], which capture progress over time. The primary goal is to advance the state of the art on a common problem rather than to declare a winner, although a scoring system exists and the organizers execute submitted code independently.

The PACE Challenges [31, 41] also address classical combinatorial problems, such as hitting set, but operate as formal competitions with live leaderboards. They feature two tracks: exact solvers, scored by the number of solved instances, and heuristic solvers, scored by a function mapping each solution to $[0, 1]$ based on the best-known solution and instance properties. A junior category is also offered. All implementations are executed by the organizers under strict resource constraints, most notably a single-thread restriction, with half of the instances kept secret. The competition typically runs for about nine months.

The ROADEF Challenges [3], organized since 1999 by the French Operations Research & Decision Support Society, are more applied. An industrial partner provides a challenging optimization problem and corresponding instances. Some problems include strong geometric components, such as the 2018 challenge, where cutting patterns had to be optimized to minimize glass waste. These problems involve complex additional constraints and objectives, distinguishing them from the academic variants used in DIMACS and PACE. Only solution quality under different resource constraints is evaluated: no limits in the first phase, followed by two distinct limits in the finals. Scoring uses a Borda-count system, ranking solutions and assigning points accordingly[1]. There is no live leaderboard, but multiple qualification rounds with published results may occur. The competition typically lasts more than a year.

[1] In the following, we refer to any scoring system that assigns scores based on the rank of solutions, that is, identifying the best solution, second-best solution, and so forth, as *Borda* style. Such scores may be aggregated either as the mean rank or by assigning points to each rank and summing them.

The International Timetabling Competitions [29] have a similar character but impose no resource limitations and accept any valid solution. Notably, they use Instance Space Analysis for instance generation. Unlike our approach, which selects a diverse subset from a large pool of randomly generated, highly correlated instances, they generate diverse synthetic instances from a limited set of real instances.

The League of Robot Runners Competitions [10] addresses a problem similar to *CG:SHOP 2021* [36], where robots must navigate a grid while avoiding obstacles and other robots, but with notable differences in problem definition and organization. In the 2024 edition, the problem comprised two components: task assignment and collision-free navigation. Participants could solve only one of these components, with a default implementation handling the other, or both, resulting in three tracks. A strong online component was introduced, as tasks were revealed only during execution and integrated into the real-time evaluation conducted by the organizers, where slow decisions caused costly idle steps, though solvers were granted 30 minutes of preparation per map. Unlike our competition, submissions had to be open source and a live leaderboard was maintained.

The VeRoLog [28] competitions, which address variants of the Vehicle Routing Problem, include both an unconstrained track, allowing participants to use their own computational resources, and a constrained track with fixed per-instance runtimes.

The DISPLIB Challenge [6] is notable for using an honor-based resource limit, also adopted in the Delivery Hero Challenge [15], part of CO@Work 2024. The latter further restricted excessive computation by releasing the final instances only shortly before the deadline.

The Kaggle Christmas Challenges [9] and Google Hash Code [2] feature playful combinatorial optimization problems aimed beyond the research community, attracting large participant numbers and using live leaderboards. Scores correspond directly to objective values, and no resource limits are imposed; only solution quality matters.

Finally, the SAT Competitions [32, 20] and the MiniZinc Challenge [46, 12] function more as controlled benchmarks than traditional challenges, tracking the state of the art in SAT and CP solvers, respectively. Consequently, they pose a high entry barrier for newcomers, and the leaderboards are typically dominated by the same research groups.

In Table 1, we summarize the characteristics of the competitions discussed above.

Competition	Character	Metric	Scoring	Resource Limits	Submission	Timeline	Leaderboard
CG:SHOP	Academic	Quality	Rel. to best	No	Solutions	Months	No
PACE Exact	Academic	Proven opt.	Absolute	Yes	Executables	$\sim$9 months	Live
PACE Heuristic	Academic	Quality	Rel. to best	Yes	Executables	$\sim$9 months	Live
ROADEF/EURO	Applied	Quality	Borda	Hybrid	Hybrid	>1 year	Phase results
ITC	Applied	Quality	Borda	No	Solutions	>1 year	Phase results
VeRoLog All-Time-Best	Applied	Quality	Borda	No	Solutions	Months	No
VeRoLog Restricted	Applied	Quality	Borda	Yes (inst.-spec.)	Executables	Months	No
Google Hash Code	Fun	Quality	Absolute	No	Solutions	Hours	Live
Kaggle Santa	Fun	Quality	Absolute	No	Solutions	Weeks	Live
DISPLIB	Applied	Quality	Borda	Yes (honor-based)	Solutions	Months	Phase results
CO@Work 2024	Applied (simplified)	Quality	Borda	Yes (honor-based)	Solutions	Days	No
Robot Runners	Academic	Quality	Rel. to best	Yes	Executables	Months	Live

Table 1 Comparison of selected competitions. DIMACS, SAT, and MiniZinc are excluded due to their distinct characteristics and evaluation criteria.

For the *character* category, we distinguish three types: academic, applied, and fun competitions. Academic competitions focus on abstract problems, as in theoretical computer science, and are typically organized by researchers for researchers. These problems are often simpler in formulation than real-world applications, making them more approachable. Applied competitions address more complex, real-world problems, often (co-)organized by industry practitioners or professional bodies. Problem descriptions may be simplified for accessibility, especially for students. Fun competitions are designed primarily as challenges without scientific or practical objectives. This classification is not always clear-cut; for instance, the Robot Runners competition addresses a practically relevant topic but frames the challenge at an abstract level using artificial instances, similar to those used in academic research papers. CG:SHOP falls clearly into the academic class, as while some problems actually have practical relevance, e.g., the packing problem in 2024, this is not a mandatory feature and instances are not derived from real-world applications.

Regarding the *metric*, most competitions evaluate only solution quality, as it is the simplest to measure. PACE also has an exact track; however, because optimality proofs are difficult to produce and verify, they rely on disqualifying participants found to violate the rules or to produce any incorrect answers. The SAT Competitions require infeasibility proofs, which can be large and complex but are standardized in the community. In the MiniZinc Challenges, evaluation is lexicographic: feasibility first, then objective value, and finally runtime. CG:SHOP evaluates only solution quality, with submission time considered only as a potential tie-breaker, although this has never been required. The first installment in 2019 also featured an optimality track, but participation was too low for it to be continued.

For *scoring*, most competitions use a Borda count, ranking solutions for each instance and assigning points accordingly. Variants include awarding points only to the top-k entries or decreasing points more steeply for lower ranks. Multi-phase competitions may assign points at the end of each phase, with later phases weighted more heavily. When metrics are on comparable scales, scores may be summed directly; otherwise, objective values are normalized to $[0, 1]$ using instance properties and the best-known solution. This scaling can be non-linear; for example, PACE Heuristic uses both the best and a baseline solution to define a rewarded range with a non-linear transition. CG:SHOP adopts a similar scaling approach, which we describe in more detail in Section 4.2.

Resource limits vary. Some competitions impose none. Others use honor-based constraints, specifying maximum time and cores per instance without enforcement. In CO@Work 2024, this was reinforced by a short window between instance release and submission deadline. Some, such as ROADEF, enforce strict limits only for finalists; in ROADEF, solvers are tested under two time limits, with higher weight on the shorter one. Others execute all solutions on dedicated hardware, with either uniform or instance-specific constraints. At CG:SHOP, no resource limits are currently enforced. This has occasionally led to concerns, as some winning teams had access to large computing clusters. However, in our assessment, their success cannot be attributed to computational resources alone but also to algorithmic insights.

If limits are enforced, participants must *submit* executables or solver source code; otherwise, solution files suffice. Some competitions, such as the League of Robot Runners also require the source code to be released under an open-source license, potentially also restricting the use of proprietary optimization software common in optimization.

Competition *timelines* are usually several months, except for event-driven formats like Google Hash Code or CO@Work 2024. Longer competitions may have multiple phases with distinct deadlines; others evaluate only at the end. The first installment of CG:SHOP in 2019 lasted only one month, which proved too short; since then, the competition has typically spanned several months, with the exact duration determined by external factors. *Leaderboard* policies vary: some are live, others publish phase results, and some reveal final standings only at the end—sometimes with substantial delays, e.g., at a conference. At CG:SHOP, no public live leaderboard is maintained, although in some editions participants were able to view their own scores.

Except for the Kaggle Santa competitions, all competitions usually have tens to a few hundred instances. A few have junior categories, although this seems to be relatively rare. Frequently, especially for non-regular competitions, submission is done via email and only a few competitions offer online submission forms.

3 Server architecture

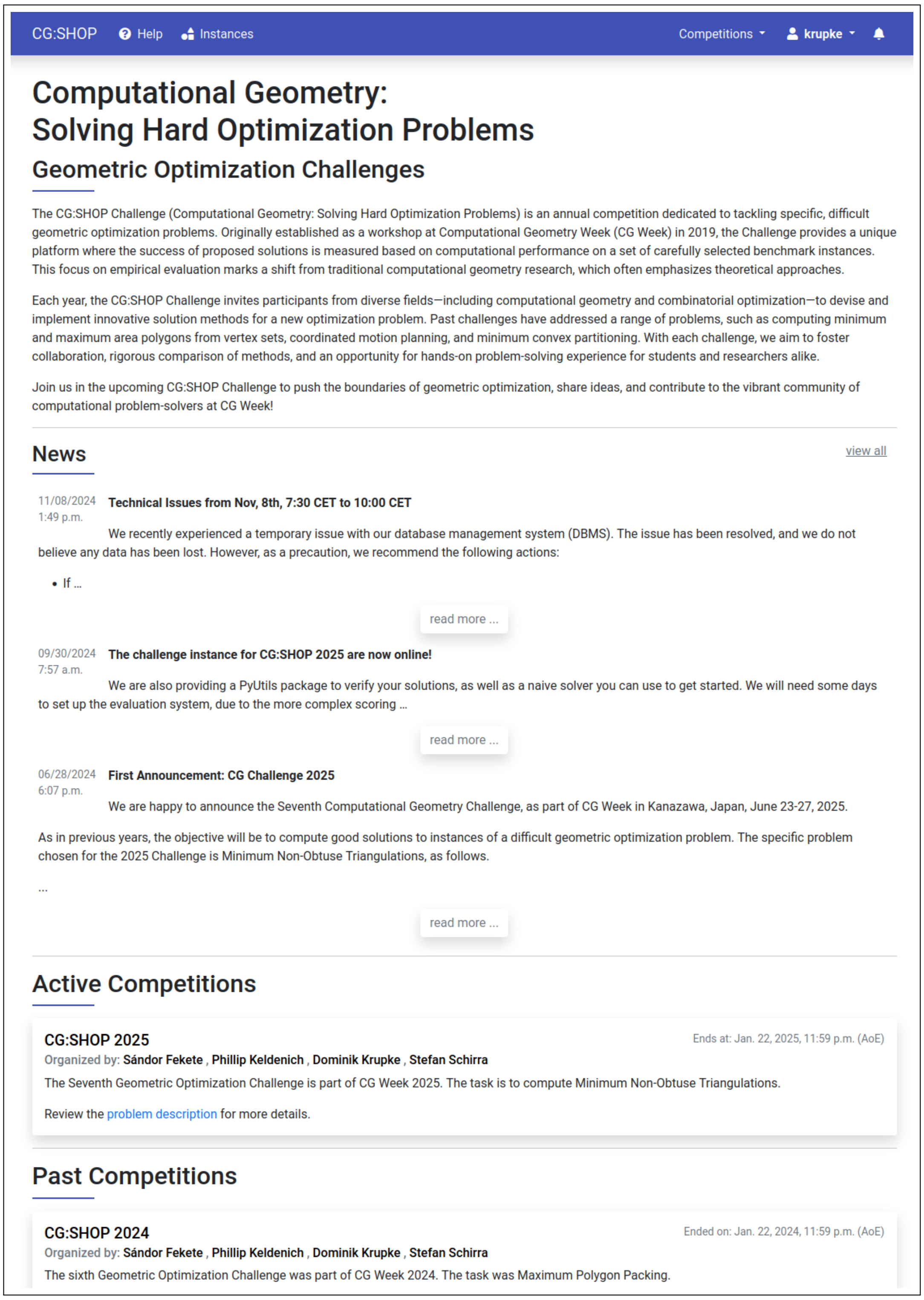
The CG:SHOP competitions are managed via a dedicated website and server architecture. The main page provides an overview of announced, ongoing, and past competitions, along with recent updates, as shown in Figure 1. Selecting a competition displays detailed competition-specific information and options, as seen in Figure 2. After users logged in, they can upload a submission consisting of solutions to one or more of the competition's instances on behalf of one of their teams. Once a submission is uploaded, its verification is distributed to a Slurm cluster, which processes the task and, upon completion, updates the database with the result and notifies the team, potentially including an explanation for any rejections. The following sections outline the frameworks and libraries used, as well as the components developed for this system.

Generic competition platforms, such as `https://www.kaggle.com`, `https://optil.io/`, and `https://www.codabench.org/`, could in principle have been used. However, based on prior experience, we anticipated that verifying geometric data would be computationally demanding due to the use of exact arithmetic. Developing a dedicated system gave us greater control over the verification process. Kadıoğlu and Kleynhans [44] likewise built a custom competition platform and discuss their design choices. A comprehensive survey of *online judge systems* is provided by Wasik et al. [56], who classify platforms by application area, describe common evaluation procedures, and highlight their use in contexts ranging from competitive programming and education to industrial optimization challenges. Most of the surveyed systems focus on the challenge of executing participant code automatically in secure, sandboxed environments with strict control over resource limits such as time and memory. This issue is not relevant for CG:SHOP, as we chose not to impose strict resource limits and instead allowed participants to run their code on their own machines, under the principle that algorithmic ideas matter more than computational power, thereby simplifying this aspect for both participants and organizers.

3.1 Frameworks and libraries

The website relies on a selection of frameworks and libraries deployed across several servers, as shown in Figure 3. This approach simplifies development while ensuring security and reliability. The architecture comprises three machine types: the web server hosting the website (deployed with Docker containers), a monitoring server to detect web server failures, and a cluster of verification workers for processing submissions. Not explicitly shown is the Network File System (NFS) server, which is part of the infrastructure and provides daily backups of all components. Below, we describe the essential software/framework choices and the rationale for each.

Python [19] Selected as the primary language due to its versatility and prevalence among



Computational Geometry:
Solving Hard Optimization Problems

Geometric Optimization Challenges

The CG:SHOP Challenge (Computational Geometry: Solving Hard Optimization Problems) is an annual competition dedicated to tackling specific, difficult geometric optimization problems. Originally established as a workshop at Computational Geometry Week (CG Week) in 2019, the Challenge provides a unique platform where the success of proposed solutions is measured based on computational performance on a set of carefully selected benchmark instances. This focus on empirical evaluation marks a shift from traditional computational geometry research, which often emphasizes theoretical approaches.

Each year, the CG:SHOP Challenge invites participants from diverse fields—including computational geometry and combinatorial optimization—to devise and implement innovative solution methods for a new optimization problem. Past challenges have addressed a range of problems, such as computing minimum and maximum area polygons from vertex sets, coordinated motion planning, and minimum convex partitioning. With each challenge, we aim to foster collaboration, rigorous comparison of methods, and an opportunity for hands-on problem-solving experience for students and researchers alike.

Join us in the upcoming CG:SHOP Challenge to push the boundaries of geometric optimization, share ideas, and contribute to the vibrant community of computational problem-solvers at CG Week!

News

view all

11/08/2024 1:49 p.m. **Technical Issues from Nov, 8th, 7:30 CET to 10:00 CET**

We recently experienced a temporary issue with our database management system (DBMS). The issue has been resolved, and we do not believe any data has been lost. However, as a precaution, we recommend the following actions:

- If ...

read more ...

09/30/2024 7:57 a.m. **The challenge instance for CG:SHOP 2025 are now online!**

We are also providing a PyUtils package to verify your solutions, as well as a naive solver you can use to get started. We will need some days to set up the evaluation system, due to the more complex scoring ...

read more ...

06/28/2024 6:07 p.m. **First Announcement: CG Challenge 2025**

We are happy to announce the Seventh Computational Geometry Challenge, as part of CG Week in Kanazawa, Japan, June 23-27, 2025.

As in previous years, the objective will be to compute good solutions to instances of a difficult geometric optimization problem. The specific problem chosen for the 2025 Challenge is Minimum Non-Obtuse Triangulations, as follows.

...

read more ...

Active Competitions

CG:SHOP 2025

Ends at: Jan. 22, 2025, 11:59 p.m. (AoE)

Organized by: **Sándor Fekete** , **Phillip Keldenich** , **Dominik Krupke** , **Stefan Schirra**

The Seventh Geometric Optimization Challenge is part of CG Week 2025. The task is to compute Minimum Non-Obtuse Triangulations.

Review the problem description for more details.

Past Competitions

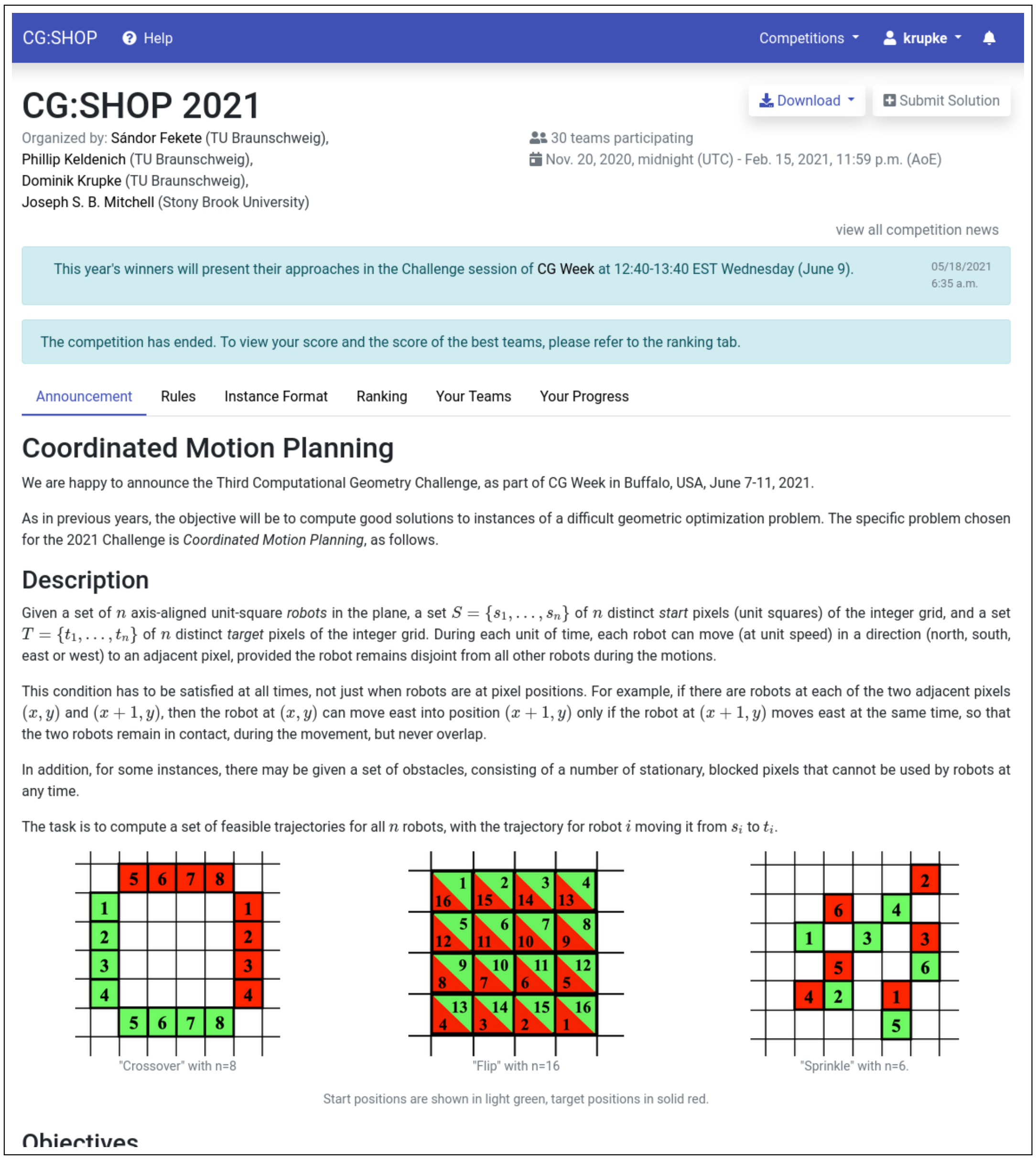
CG:SHOP 2024

Ended on: Jan. 22, 2024, 11:59 p.m. (AoE)

Organized by: **Sándor Fekete** , **Phillip Keldenich** , **Dominik Krupke** , **Stefan Schirra**

The sixth Geometric Optimization Challenge was part of CG Week 2024. The task was Maximum Polygon Packing.

Figure 1 The main page `https://cgshop.ibr.cs.tu-bs.de/` provides an overview of competitions and recent news.



CG:SHOP 2021

Organized by: Sándor Fekete (TU Braunschweig), Phillip Keldenich (TU Braunschweig), Dominik Krupke (TU Braunschweig), Joseph S. B. Mitchell (Stony Brook University)

30 teams participating

Nov. 20, 2020, midnight (UTC) - Feb. 15, 2021, 11:59 p.m. (AoE)

view all competition news

This year's winners will present their approaches in the Challenge session of CG Week at 12:40-13:40 EST Wednesday (June 9). 05/18/2021 6:35 a.m.

The competition has ended. To view your score and the score of the best teams, please refer to the ranking tab.

Coordinated Motion Planning

We are happy to announce the Third Computational Geometry Challenge, as part of CG Week in Buffalo, USA, June 7-11, 2021.

As in previous years, the objective will be to compute good solutions to instances of a difficult geometric optimization problem. The specific problem chosen for the 2021 Challenge is *Coordinated Motion Planning*, as follows.

Description

Given a set of n axis-aligned unit-square *robots* in the plane, a set $S = \{s_1, \ldots, s_n\}$ of n distinct *start* pixels (unit squares) of the integer grid, and a set $T = \{t_1, \ldots, t_n\}$ of n distinct *target* pixels of the integer grid. During each unit of time, each robot can move (at unit speed) in a direction (north, south, east or west) to an adjacent pixel, provided the robot remains disjoint from all other robots during the motions.

This condition has to be satisfied at all times, not just when robots are at pixel positions. For example, if there are robots at each of the two adjacent pixels (x, y) and $(x+1, y)$, then the robot at (x, y) can move east into position $(x+1, y)$ only if the robot at $(x+1, y)$ moves east at the same time, so that the two robots remain in contact, during the movement, but never overlap.

In addition, for some instances, there may be given a set of obstacles, consisting of a number of stationary, blocked pixels that cannot be used by robots at any time.

The task is to compute a set of feasible trajectories for all n robots, with the trajectory for robot i moving it from s_i to t_i.

"Crossover" with n=8

"Flip" with n=16

"Sprinkle" with n=6.

Start positions are shown in light green, target positions in solid red.

Objectives

Figure 2 The competition page offers an interface to access relevant information and submit solutions.

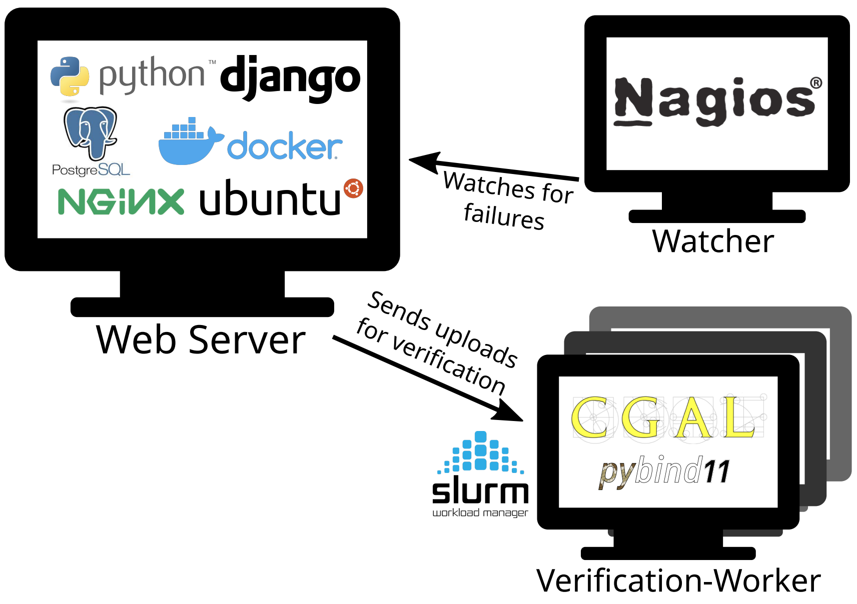


Figure 3 The server architecture consists of a web server, a monitoring server, and a dynamic set of verification workers. Logos are intellectual property of the respective projects. Not shown is the NFS server that provides daily backups.

students, making it ideal for training assistants. Python's support for C/C++ integration allows for efficient handling of complex tasks.

Django [7] The Django web framework features functionalities, such as database management, user authentication, and security, as well as a robust template system. Its community support and comprehensive tutorials were additional advantages, making it accessible for students with basic Python experience.

Ubuntu Linux LTS [27] The server uses Ubuntu Linux with long-term support to ensure stability and minimize software changes, as frequent feature updates are unnecessary for stable and secure operation.

Docker [8] Docker containers facilitate simple setup and reliable composition of the different components, such as PostgreSQL and Nginx, supporting consistent replication for testing and debugging while insulating critical components from OS-level changes.

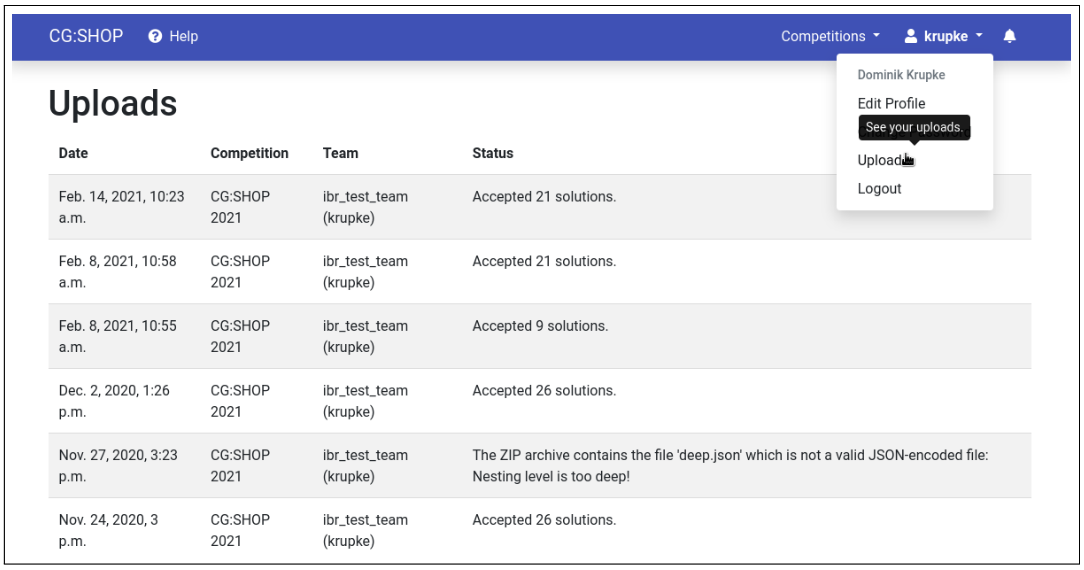
PostgreSQL [17] PostgreSQL is used as the relational database, managed primarily through Django, and deployed in a Docker container for stability. Other well-supported databases, such as MariaDB, would also be suitable.

Nginx [14] While Django includes a development server, Nginx was selected for production due to its speed and support for HTTPS. It offers a well-documented integration with Django and allows integrating further services, such as the instance database, under the same domain despite being a completely separated project.

CGAL [54] Exact arithmetic is critical in submission verification to avoid numeric issues common in geometric problems. CGAL provides the necessary precision and efficiency, although it requires the core components of verifiers to be written in C++.

pybind11 [43] Using pybind11, we integrate the C++ verifiers into our Python system. We developed the Python tool `skbuild-conan` [24] to easily build and package the verifiers utilizing the Conan package manager [4] and the *scikit-build* [21] build system.

Slurm [25] Slurm, a workload manager, distributes verification tasks among multiple workers.



Uploads

Date	Competition	Team	Status
Feb. 14, 2021, 10:23 a.m.	CG:SHOP 2021	ibr_test_team (krupke)	Accepted 21 solutions.
Feb. 8, 2021, 10:58 a.m.	CG:SHOP 2021	ibr_test_team (krupke)	Accepted 21 solutions.
Feb. 8, 2021, 10:55 a.m.	CG:SHOP 2021	ibr_test_team (krupke)	Accepted 9 solutions.
Dec. 2, 2020, 1:26 p.m.	CG:SHOP 2021	ibr_test_team (krupke)	Accepted 26 solutions.
Nov. 27, 2020, 3:23 p.m.	CG:SHOP 2021	ibr_test_team (krupke)	The ZIP archive contains the file 'deep.json' which is not a valid JSON-encoded file: Nesting level is too deep!
Nov. 24, 2020, 3 p.m.	CG:SHOP 2021	ibr_test_team (krupke)	Accepted 26 solutions.

Figure 4 Users can track the status of their uploads, along with those of team members. Status messages indicate issues if present.

Initially, Celery was used but required a separate message broker. By leveraging an existing, professionally managed Slurm cluster and developing `slurminade` [26], we created a streamlined interface for Python function distribution, similar to Celery, but outsourcing the administration overhead.

Nagios [13] Nagios monitors the web server and alerts administrators of downtimes or issues, such as expiring SSL certificates. Hosted on a separate server, it ensures continuous oversight even in case of a primary server outage.

Bootstrap [1] Bootstrap enables the construction of clean, modern web interfaces, simplifying design without compromising functionality.

3.2 Components

The frameworks and libraries described above support several core components essential to the competition's operation. A brief overview of these components is given below:

User Management To handle user accounts, our system expands on Django's base user model, adding fields such as affiliation data and options for password resetting, essential given that competitions are held annually and passwords are often forgotten. This part was straightforward to implement using Django's flexible user model extensions, made accessible by community tutorials.

Team Management Recognizing that competitions emphasize team participation, we designed the system to be built around teams. Users can create and manage teams, inviting members as needed. Each team's actions, including adding or removing members, are unrestricted among members, allowing frictionless collaboration and easy management. Administrators can reverse any changes if issues arise. To preserve team structure over time, teams are competition-specific and locked after the submission deadline, ensuring that team membership and affiliation are fixed at the time of participation. While this requires participants to create a new team for each competition, it simplifies the system and reduces the risk of unintended interactions between different events.

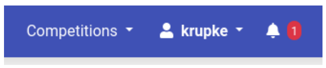


Figure 5 A red badge discreetly alerts users to new notifications.

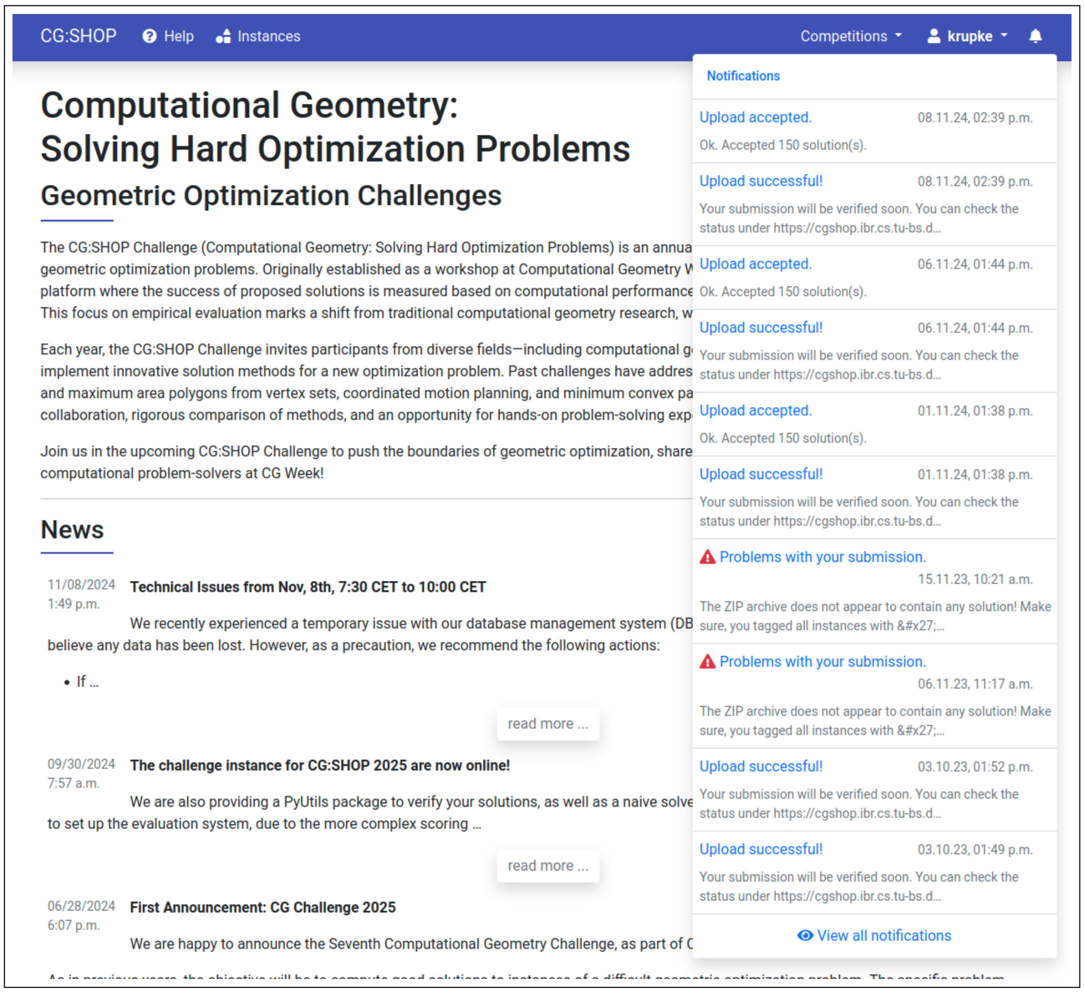
Competition Management Effective management of each competition involves handling content, timing, and specific rules, all controlled through a flexible system. Competitions are initiated by cloning the app of a prior instance, followed by customization of the competition key, verifier, and relevant details. Django's templating and app architecture facilitate this setup, allowing each competition to be isolated and modified as needed. While further generalization is possible, the current approach enables straightforward, competition-specific customization, reducing the risk of unintended interactions between different competitions. Some complex components, such as score tracking and content management systems, have been generalized and can be imported as needed, avoiding redundant implementation efforts for each new competition. The main limitations of this approach are the need for basic Django knowledge to configure a new competition and the potential for redundant migrations if there are breaking changes in dependencies; however, these issues are minor given the infrequency of new competition setups. Since competition requirements can vary significantly, the current strategy provides a balanced approach between flexibility and simplicity, allowing adjustments as we gain insights from each event.

Submission System The submission system is designed to be adaptable for each competition by simply updating the competition key and adding a unique verifier. Each submission is uploaded as a zip file, verified by a competition-specific verifier tool. Once uploaded, the system notifies the verification handler of the corresponding competition, and users can view the status of their submissions within their accounts, as shown in Figure 4. This modularity simplifies adaptation for new competitions and maintains consistency across events. We also consider the transparency of the status of the submissions as a critical aspect, as participants often spent a significant amount of time on their solutions, and we want to assure them that their submissions are being processed properly.

Notification System Effective communication with participants is critical. The notification system provides subtle alerts via a red badge, as seen in Figure 5. Clicking the badge opens the notification list, as in Figure 6, where users can view detailed messages. This approach allows direct feedback on actions, such as submission status, without relying on excessive emails, maintaining a balance between user engagement and system transparency.

News System Major announcements are distributed through community mailing lists to reach a wide audience. Smaller updates, such as maintenance notices or minor corrections, are posted through an integrated blog-like news section, providing participants with timely and relevant information without flooding inboxes. Frequent news items on the webpage also indicate the active maintenance of the system.

Admin System Django includes a robust admin interface, which we have extended to facilitate an organized overview of competition data. This extension also allows for quick querying and downloading of data, such as individual submission files, for review purposes. Moreover, users with admin status have unrestricted access to all teams' scores and progress on the actual competition side, as if they were team members, enabling seamless monitoring of participation. It also allows entering submissions on behalf of participants even after the deadline, in case of technical issues or other emergencies. This feature already had to be used in the past, and it is reassuring as an organizer to have this option



Computational Geometry:
Solving Hard Optimization Problems

Geometric Optimization Challenges

The CG:SHOP Challenge (Computational Geometry: Solving Hard Optimization Problems) is an annua
geometric optimization problems. Originally established as a workshop at Computational Geometry W
platform where the success of proposed solutions is measured based on computational performance
This focus on empirical evaluation marks a shift from traditional computational geometry research, w

Each year, the CG:SHOP Challenge invites participants from diverse fields—including computational g
implement innovative solution methods for a new optimization problem. Past challenges have addres
and maximum area polygons from vertex sets, coordinated motion planning, and minimum convex pa
collaboration, rigorous comparison of methods, and an opportunity for hands-on problem-solving exp

Join us in the upcoming CG:SHOP Challenge to push the boundaries of geometric optimization, share
computational problem-solvers at CG Week!

News

11/08/2024 1:49 p.m. **Technical Issues from Nov, 8th, 7:30 CET to 10:00 CET**

We recently experienced a temporary issue with our database management system (DB
believe any data has been lost. However, as a precaution, we recommend the following actions:

- If ...

read more ...

09/30/2024 7:57 a.m. **The challenge instance for CG:SHOP 2025 are now online!**

We are also providing a PyUtils package to verify your solutions, as well as a naive solve
to set up the evaluation system, due to the more complex scoring ...

read more ...

06/28/2024 6:07 p.m. **First Announcement: CG Challenge 2025**

We are happy to announce the Seventh Computational Geometry Challenge, as part of C

As in previous years, the objective will be to compute good solutions to instances of a difficult geometric optimization problem. The specific problem

Notifications

Upload accepted. 08.11.24, 02:39 p.m.
Ok. Accepted 150 solution(s).

Upload successful! 08.11.24, 02:39 p.m.
Your submission will be verified soon. You can check the status under https://cgshop.ibr.cs.tu-bs.d...

Upload accepted. 06.11.24, 01:44 p.m.
Ok. Accepted 150 solution(s).

Upload successful! 06.11.24, 01:44 p.m.
Your submission will be verified soon. You can check the status under https://cgshop.ibr.cs.tu-bs.d...

Upload accepted. 01.11.24, 01:38 p.m.
Ok. Accepted 150 solution(s).

Upload successful! 01.11.24, 01:38 p.m.
Your submission will be verified soon. You can check the status under https://cgshop.ibr.cs.tu-bs.d...

Problems with your submission. 15.11.23, 10:21 a.m.
The ZIP archive does not appear to contain any solution! Make sure, you tagged all instances with &#x27;...

Problems with your submission. 06.11.23, 11:17 a.m.
The ZIP archive does not appear to contain any solution! Make sure, you tagged all instances with &#x27;...

Upload successful! 03.10.23, 01:52 p.m.
Your submission will be verified soon. You can check the status under https://cgshop.ibr.cs.tu-bs.d...

Upload successful! 03.10.23, 01:49 p.m.
Your submission will be verified soon. You can check the status under https://cgshop.ibr.cs.tu-bs.d...

View all notifications

Figure 6 Detailed notification list for user interactions.

easily available without having to resort to low level database manipulation.

Solution Management Managing solutions and their corresponding scores in a database poses several challenges, particularly in ensuring efficient access and updates. Scalability is a critical concern, as some competitions generate tens of thousands of submissions due to participants automatically submitting newly discovered solutions. The database must efficiently handle frequent updates and ranking queries.

Additional complexities arise from the distributed verification process, where results may arrive out of order. The system must also accommodate the revision of submissions mistakenly rejected due to minor issues, such as ambiguities in format specifications. In rare cases, submissions may need to be deleted, necessitating careful measures to preserve database integrity.

To prevent disruptions, the system must continue processing other submissions even when unexpected issues delay individual cases. Some competitions use relative scores, which depend on the best solution and can change as better solutions are submitted. Additionally, tracking the progress of the competition requires maintaining a historical record of how scores have evolved over time.

Satisfying all these requirements simultaneously is highly challenging, making solution management one of the most complex components of the system.

Verification System Each competition requires unique verification utilities, which we publish on PyPI for convenient access. Verification is resource-intensive and should not be conducted on the web server itself. Geometric problems often involve arithmetic challenges that require exact calculations, such as those provided by CGAL. However, exact arithmetic can lead to unpredictable memory consumption, potentially causing out-of-memory errors. The failure of a single verification worker should not compromise the system's stability. Although numerical issues can often be mitigated with heuristics in practical scenarios, relying on heuristics for evaluation could overlook critical edge cases and result in unfair rejection of superior solutions, which would undermine trust in the evaluation process. Given that the majority of uploads occur just before the deadline, participants expect timely results, allowing them to still address unforeseen issues with their solutions in time. Therefore, our system is built to scale the verification workers dynamically using our Slurm cluster.

4 Workload and tasks

In this section, we examine the tasks and workloads required to host the CG:SHOP competition over the first five years.

4.1 Problem selection

Before launching a competition, selecting a suitable optimization problem is essential. The problem must be challenging but not overly specialized, ideally motivated by real-world applications yet not too well-explored. For instance, the TRAVELING SALESMAN PROBLEM is unsuitable due to Concorde's dominance [30]. Fortunately, a call for problems and an experienced advisory board simplify this process, though coordinating with all members can be challenging. Problem selection should ideally be interwoven with preliminary instance generation and analysis, as even the experience of the advisory board may not suffice to estimate the problem's practical difficulty accurately. The effort required for this process can vary significantly, depending on how familiar the problem is. We also typically combined this step with discussions about the scoring function as the ability to score a solution is a crucial aspect of the problem selection.

4.2 Score

Specifying the scoring function is one of the more complex aspects of organizing a competition. Several considerations must be balanced, including clarity, a clear distinction between naive and advanced solutions, and a fair weighting across all instances.

The results of CG:SHOP 2020 illustrate the difficulty of determining a winner. One team achieved 126 uniquely best solutions, far more than any other team. Another team, however, produced solutions consistently close to the best, and on 11 instances obtained significantly better objective values than any competitor. In the end, this second team won, while the team with the most best solutions placed second.

Many competitions use Borda-style scoring, where points are assigned based on rank: the best solution receives the most points, the second-best slightly fewer, and so on. The team with the highest total score then wins. We considered this approach problematic for some of our problems, namely those with continuous objectives. For example, when computing polygons of minimal area, multiple solutions may be very close in value. This could require exact arithmetic to compare objective values and would unfairly penalize participants whose solutions are only slightly worse, since small tolerances are commonly accepted in optimization. Introducing tolerances into a rank-based system also appeared risky, as even small changes could substantially alter the final rankings, and selecting a fair tolerance value would be difficult in advance without knowing the variance of submitted solutions. We do, however, consider such scoring functions adequate for problems with relatively small integer objectives, and we may use them in future competitions.

In the first two editions, CG:SHOP 2019 and CG:SHOP 2020, we used a scoring function that scaled objective values for each solution to the interval $[0, 1]$ using theoretical bounds, with the winner determined by the sum of scores across all instances. This approach had the advantage that participants could compute their own scores independently and monitor progress without revealing the results of other teams. However, it also had serious drawbacks. Not all problems have easily accessible theoretical bounds, and when available, such bounds may be very loose, sometimes by several orders of magnitude. As a result, even optimal solutions could yield very small scores, and naive and advanced solutions would not be clearly distinguished. Moreover, bound tightness could vary substantially across instances, producing skewed and unbalanced scores that arbitrarily favored certain instances.

For all later editions, we therefore adopted a relative scoring system based on the best known solution, as also used in the PACE Heuristic track. A drawback of this approach is that scores are dynamic: they reveal information about competitors and may decrease over time as better solutions are found, which can confuse participants if scores are visible during the competition. To address this, we chose to withhold overall scores until the end, releasing them only once finalized. The simplest variant of this system is the z/z^* normalization for maximization problems, where z is the objective value of the solution and z^* the best known solution. When naive and advanced solutions differ only by a few percentage points, it can be desirable to emphasize this gap by applying a non-linear transformation to the normalized score, such as squaring it, which has become our default choice. Other transformations, such as higher powers or alternative functions, may also be considered depending on the desired emphasis. Additionally, normalization can incorporate a naive baseline when available, reducing the score of naive solutions close to zero.

No scoring function is perfect, but the most important property is transparency. A clearly specified scoring rule establishes expectations in advance and minimizes the risk of disputes. While it is always possible that the runner-up may be regarded by the community as the stronger team, as long as the scoring process is unbiased and well-defined, the outcome should be accepted.

4.3 System development

The development and maintenance of the system are among the most significant workloads and encompass three main areas: the general webpage, competition-specific components, and server maintenance and troubleshooting.

Initially, we underestimated the effort required for system development and maintenance, as the first competition's system was built rapidly. However, this initial version had several shortcomings, including a limited user experience for both participants and organizers, as well as scalability and reliability issues, some of which stemmed from infrastructure constraints.

A major challenge was designing an optimal user experience for participants while identifying the essential features required by organizers. Predictably, most participant activity occurred in the final days leading up to the deadline, with minimal engagement during the rest of the competition. As inexperienced organizers, we initially attempted to derive early insights from limited data using monitoring features, which often failed to provide meaningful results while being costly to maintain.

Another unanticipated issue was the use of automated submission processes by some participants, which overloaded the database and verification system. Addressing these scalability challenges often required schema modifications, which are particularly complex during an active competition. Such changes necessitate meticulous planning to prevent data loss or corruption.

4.3.1 Generic webpage

The initial webpage, created using Django's built-in features, handled file uploads, user management, and submission verification through a separate process monitoring new uploads and updating the database. Despite our limited web development experience, this system was completed in about two man-weeks for the CG:SHOP 2019 competition.

However, the initial system lacked flexibility for future competitions and faced scalability limitations. For the second competition, the webpage was redesigned to improve user experience and enable reusability. Additionally, the original verification system was replaced with a dynamic, distributed version to handle larger workloads. Drawing on lessons from the first year and benefiting from more preparation time, we adopted a modular structure and delegated tasks to student assistants where possible.

While anticipating an annual competition schedule, we aimed to generalize system components to simplify future development. This approach, however, led to some premature overengineering, as evolving requirements rendered certain components non-transferable or too costly to maintain. Consequently, we prioritized a streamlined design focusing on essential features and emphasized simplicity over reusability for competition-specific needs.

Over the years, several person-months have been invested in the system, with student assistants contributing about twice the effort. This substantial time investment reflects the learning curve associated with competition requirements and the novelty of web development tasks for algorithm engineers. In earlier competitions, maintenance and development efforts were supported by three student assistants. Nowadays, with the more mature state of the system, much of this work is managed by a single experienced assistant on a 20-40 hour per month contract.

4.3.2 Competition-specific components

Each competition required the development of a unique verifier, which aligned more closely with algorithm engineering than web development. Developing a basic verifier typically took an algorithm engineer one to two days, with additional time needed for public release on PyPI, including documentation and platform compatibility. To simplify dependency management, especially for CGAL, we developed the `skbuild-conan` tool [24], which has been released as open-source software.

Beyond testing, a key aspect of preparing the verifier was crafting clear and informative error messages for incorrect solutions, particularly for addressing edge cases that were initially unanticipated. On average, finalizing a user-friendly verifier required an additional one to two weeks of effort, which could be partially delegated to student assistants. While keeping verifiers private might have reduced the workload, we believe that making them publicly accessible enhances transparency and reusability, benefiting both participants and the broader community. Additionally, we found it effective to include simple guard mechanisms that alerted us to unexpected cases, enabling quick fixes as issues arose rather than over-investing in exhaustive pre-release testing.

4.3.3 Maintenance and troubleshooting

Server maintenance is one of the most demanding aspects of hosting CG:SHOP with issues often arising unexpectedly, as numerous external factors can introduce complications. The system consists of multiple interdependent components that require meticulous configuration to ensure seamless operation over extended periods and under significant load. Early difficulties included breaking changes from software updates, race conditions during reboots, and peculiarities related to the university's infrastructure. To address these challenges, we transitioned to Docker containers for improved environment isolation and replaced our self-managed Celery system with a professionally managed Slurm cluster, significantly enhancing stability and reliability.

Server maintenance requires an estimated three weeks of effort per year, as our custom system demands ongoing management. Over time, breaking changes in software and frameworks, coupled with security concerns, render a set-and-forget approach infeasible. Regular updates and vigilant oversight are essential to maintaining the system's security and reliability over the years.

4.4 Instance selection

Instance selection for benchmarking was relatively straightforward in the first two competitions, as point sets could be taken from existing data. However, in the second competition, we observed that collinear points significantly influenced objectives, leading to closely clustered scores for the instances with mostly low collinearity. To increase competitiveness, we introduced an additional set with many collinear points, though this late adjustment received mixed feedback.

For the third competition, which focused on parallel motion planning, we developed new instance generators designed to produce scenarios with diverse maneuvers and bottlenecks. Selecting a representative set of instances proved challenging due to the wide parameter space. Our general methodology for this process is described in detail in Section 5. While instance selection took under a day for the first two competitions, it required around two weeks for the third.

The fourth competition demanded even more time, as many generated instances turned out to be unexpectedly easy to solve during our preliminary evaluation, requiring a more meticulous selection process. After this stage, our accumulated experience and improved tools helped streamline aspects of instance selection of the later competitions. Using a logarithmic distribution of instances sizes proved effective to ensure to have a few tie breaker instances, without knowing the difficulty of the instances in advance. The primary remaining challenge is in developing the set of instance generators, in particular in allowing integer arithmetic as much as possible to not make the arithmetic the bottleneck of the competition.

4.5 Support

Throughout the competition, participant support is provided via a dedicated email address. We typically receive between 10 and 30 inquiries per competition. During the first two competitions, most issues were minor and related to webpage functionality, which we subsequently optimized. Many emails also requested clarification on competition rules or webpage features.

The most complex support requests involved investigating why certain submissions were rejected, which led us to make the verifier publicly accessible after the first competition. Rejections often stemmed from geometric edge cases or misinterpretations of the rules. Occasionally, security mechanisms – such as those guarding against malicious archives – would block files, requiring adjustments or manual overrides. This type of support initially required around 5 days per competition, but the time decreased to only a few hours as our experience grew and we improved the webpage and error messages.

4.6 Analysis

To provide more than just a ranking of scores, detailed data analysis is essential. This involves examining the instances and the problem itself, with a focus on classifying instances to identify the strengths and weaknesses of submitted solutions. Such analysis is valuable not only at the competition's conclusion but also during the event to identify issues early on. For instance, in CG:SHOP 2020, we observed that the initial instances were too simple, resulting in nearly identical submissions from the leading teams. While the first competition was straightforward to analyze, the subsequent ones required multiple days for both intermediate and final analyses.

5 Data-driven instance selection methodology

A good instance set for CG:SHOP should satisfy three key properties. First, it should cover a wide range of difficulty levels, offering both accessible instances that provide early successes and progressively harder ones that reward more advanced techniques. Second, it is essential that the set contain at least one tie-breaker instance, ensuring that the winner can be unambiguously determined. Such an instance must strike a careful balance: if it is too easy, several teams may reach optimal solutions and no ranking signal is obtained; if it is too hard, no team produces a meaningful solution, which is equally uninformative. Third, the set must remain small enough to be manageable for all participants, including students with limited computational resources.

Constructing such a set is challenging because instance difficulty is difficult to predict. Naive asymptotic analysis suggests that larger instances are harder, but in practice size alone is a poor predictor. For example, the NP-hard Minimum-Weight Triangulation (MWT) problem can be solved to optimality for instances with 30 million points in only minutes [42]. Difficulty increases substantially only when specific structural features are present, such as almost regular n-gons with a central point. Simply scaling size may eventually produce hard instances, but often for the wrong reasons: difficulty may stem less from the structure of the solution space and more from the resource demands of handling massive data. Thus, instance generation cannot rely solely on size and must also consider structural properties, although these are usually unknown in advance for new problems.

One conceivable response is to generate a huge pool of instances with many generators and rely on sheer quantity to cover the desired spectrum of difficulties and structures. However, the third property rules this out: an excessively large set would overwhelm participants with limited resources. We therefore restrict the final set to a size that remains feasible for students, estimating that around 200 instances can be solved on a single machine within the last week of the competition, assuming 30 min to 60 min of optimization per instance. This constraint makes it impossible to rely on mass alone and forces us to carefully select a compact but diverse subset.

To address this, our methodology focuses on maximizing diversity under a cardinality constraint. By selecting a structurally diverse subset from a larger pool of candidate instances, we increase the likelihood that the final benchmark covers a broad spectrum of difficulties and structures, even if their precise properties cannot be predicted. Some uninformative instances will inevitably be included, but they do little harm as long as enough interesting ones are present.

In the following, we outline the methodology used in CG:SHOP to construct such diverse instance sets. The process involves generating a large pool of instances, extracting features, defining a feature-based distance metric, and selecting a diverse subset according to this metric. Although this provides a general framework, its application always requires trial-and-error and domain-specific adjustment.

1. Implement multiple generators for the problem, each with adjustable parameters to create instances of varying size and complexity. Although the generators do not need to be perfect, they should be able to produce a large pool of random instances. It is beneficial to save the parameters used for each instance, as these may later serve as relevant features. When possible, sample from real-world data to capture structural patterns that might be difficult to obtain solely from random generation. See [36] for an example and further details on such generators.
2. Create a large pool of instances by randomly adjusting parameters to cover a broad spectrum of possibilities. Over-representing certain instance types is acceptable, as a diverse subset will be selected from this pool, ignoring overly similar instances.
3. Extract general features for each instance. For example, in a point set, features could include the total number of points, the count of collinear points, and the number of points on the convex hull, among others.
4. If feasible, implement basic heuristics or integer/constraint programming approaches for the problem (or a closely related one). Run these on the generated instances, optionally with a short time limit, and record the outcomes. For randomized approaches, running multiple trials is recommended to capture variance, which can reflect different local optima and can correlate with instance difficulty. For exhaustive search methods, record optimality gaps and their integrals (cf. *primal integral* [33]) as valuable features. Note that absolute objective values may have limited relevance due to random scaling, but relative deviations from the best or average solution can offer meaningful insights. If the performance of different heuristics varies across instances, it indicates diversity in the instances, potentially requiring distinct strategies.
5. When a feature is more meaningful in relative terms than in absolute terms, apply a logarithmic transformation to the feature. This approach is particularly suitable for features like instance size, where also a logarithmic distribution may be desirable. Similarly, for certain features, consider applying transformations such as squaring – for example, the width of a container – if the problem's difficulty is hypothesized to correlate with the feature's derived value, such as the container's area.
6. For some instances, certain features may be unavailable. Choose whether to fill in missing values – e.g., by interpolation from other instances – or to categorize the pool by generator type and treat each category separately, selecting a specific number of instances from each class.
7. Analyze feature distributions and correlations using visualizations such as pair plots, as shown in Figure 7. These plots can uncover non-linear relationships and help identify redundant or highly correlated features that can be removed to simplify the feature space. While subsequent steps, such as principal component analysis, will automatically handle

linear correlations, non-linear correlations must be addressed beforehand. Additionally, pair plots can reveal potential issues in the data, prompting a reevaluation of the selected features or adjustments to the instance generation process.

8. Scale all features to a similar range using methods like standard scaling (mean of zero, standard deviation of one) or min-max scaling (range 0 to 1).
9. Use principal component analysis (PCA) to reduce dimensionality and remove correlations among features. The optimal number of dimensions depends on the structure of the features and the acceptable error margin; however, three or four dimensions have consistently performed well in all our competitions to date. Correlations are particularly prevalent in instance features, and failing to address them can result in highly correlated features dominating the metric. This, in turn, reduces diversity in the selected instances by overshadowing variation in other features.
10. Use *kMeans* clustering in the reduced space to group instances. Set the number of clusters to match the desired instance count, and select one instance per cluster – either randomly or by choosing the cluster center – to ensure diversity. The rationale is that instances within the same cluster are similar; thus, selecting a single representative from each cluster promotes variety. Given that the distance metric is imperfect, it may be preferable to select a random instance from each cluster rather than the center, thereby introducing controlled randomness. Alternative approaches include other clustering algorithms or even actively maximizing the diversity metric via mixed integer programming, as done in MIPLIB [40]. We experimented with several methods and ultimately adopted this approach, as it appeared most effective according to our subjective assessment.
11. Assess whether the selected instances appear diverse and representative of the entire pool. It is acceptable to include some uninteresting instances, provided the set also contains a sufficient number of challenging ones. Ensuring instances are well-distributed across the full range of features increases the likelihood of achieving this balance, even without a clear definition of what constitutes a challenging instance. If the results are unsatisfactory, reevaluate the features—either create new ones or transform existing ones (e.g., using logarithmic or squared transformations). If the set is nearly satisfactory but falls short in specific aspects, you can employ a manual workaround by partitioning the instance pool into a few targeted categories and applying the selection process separately to each category, ensuring a specific number of instances is selected from each. While effective, this approach is less elegant and may not fully leverage the benefits of a unified selection process.

Useful Python tools for these steps include:

- *pandas* [16] for data handling,
- *scikit-learn* [22] for scaling and dimensionality reduction,
- *seaborn* [23] for visualizations with *matplotlib* [11], and
- *jupyter* [18] for interactive parameter tuning and documentation.

This approach draws from the work of Smith-Miles et al. [53, 52, 51, 50, 49, 47, 48] on instance feature spaces for algorithm performance prediction and the ALGORITHM SELECTION PROBLEM [45]. Their methodology uses evolutionary algorithms to generate diverse instance sets, such as in GRAPH COLORING [48], focusing on evolving instances to explore the strengths and weaknesses of fixed, existing algorithms. In contrast, our work aims to create a fixed benchmark for a challenge, where participant algorithms will evolve to perform well on our benchmark instances, rather than adapting the instances to highlight algorithmic strengths or weaknesses.

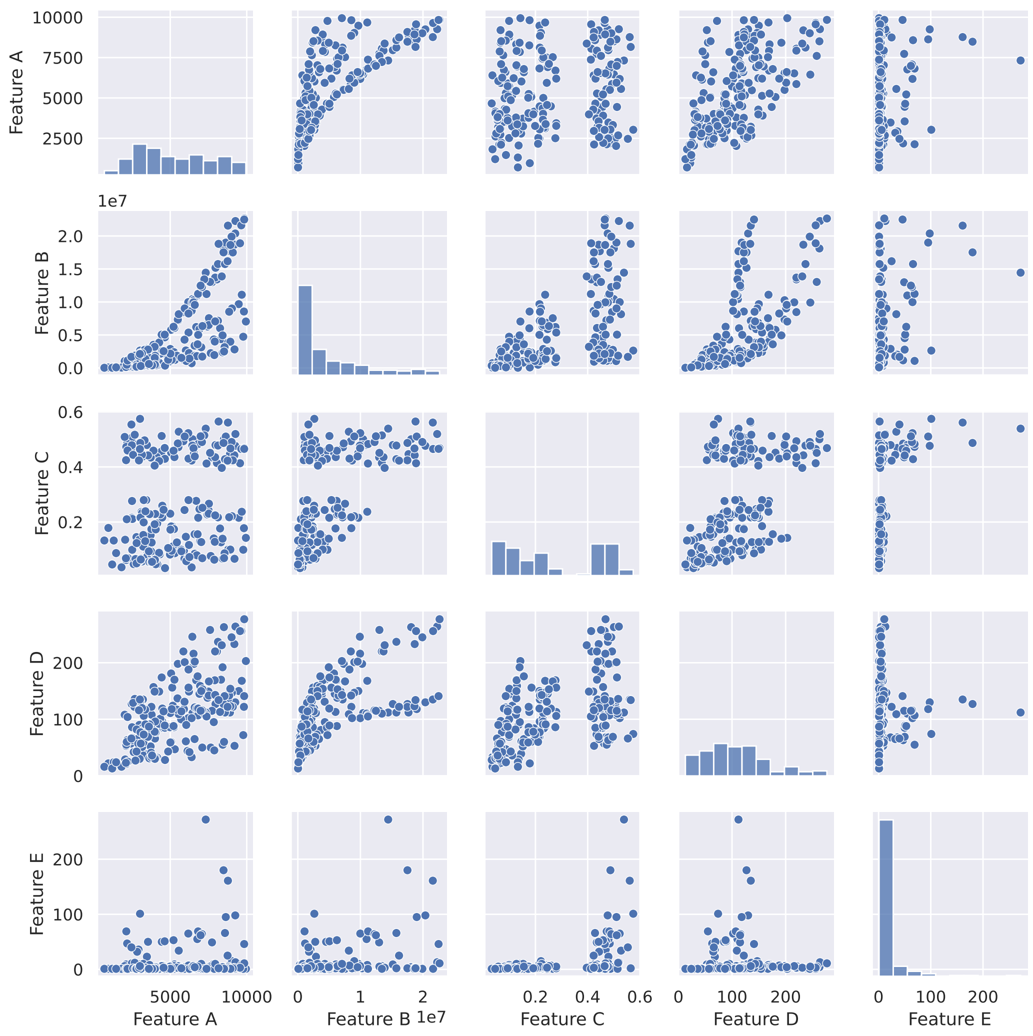


Figure 7 Pair plot of instance features from CG:SHOP 2022 [37], illustrating correlations and non-linear relationships. The data has been reduced for clarity. Notably, features A and B show a non-linear relationship, while features B and E are concentrated at lower values and feature A is uniformly distributed. Feature C exhibits a central gap, indicating the need for more instances in that range. Additionally, feature A has low correlations with features C and D, which indicate a good feature selection. Feature E includes outliers, which may represent interesting edge cases.

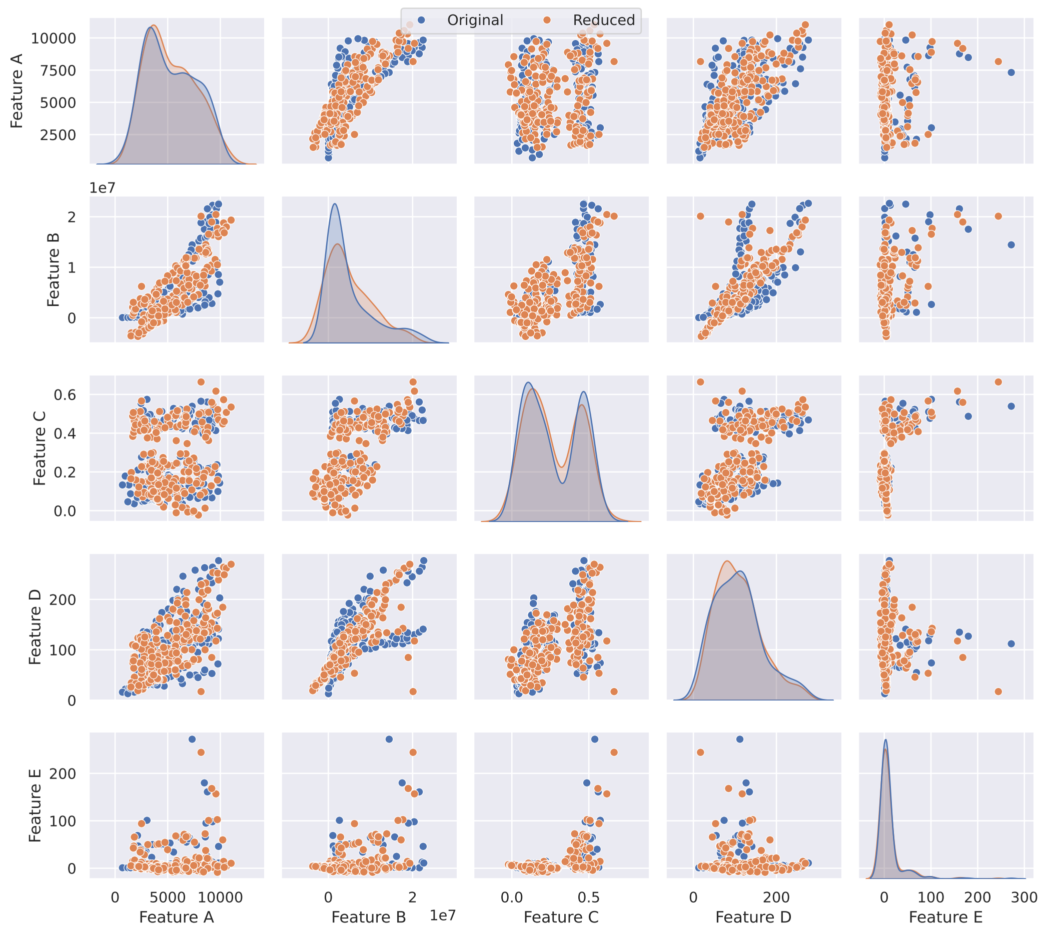


Figure 8 In this example, we reduced that data to three dimensions using standard scaling and PCA, preserving essential relationships with minimal error. The induced error is visualized by projecting the reduced data back into the original space highlighting the differences between the original (blue) and projected (orange) data points.

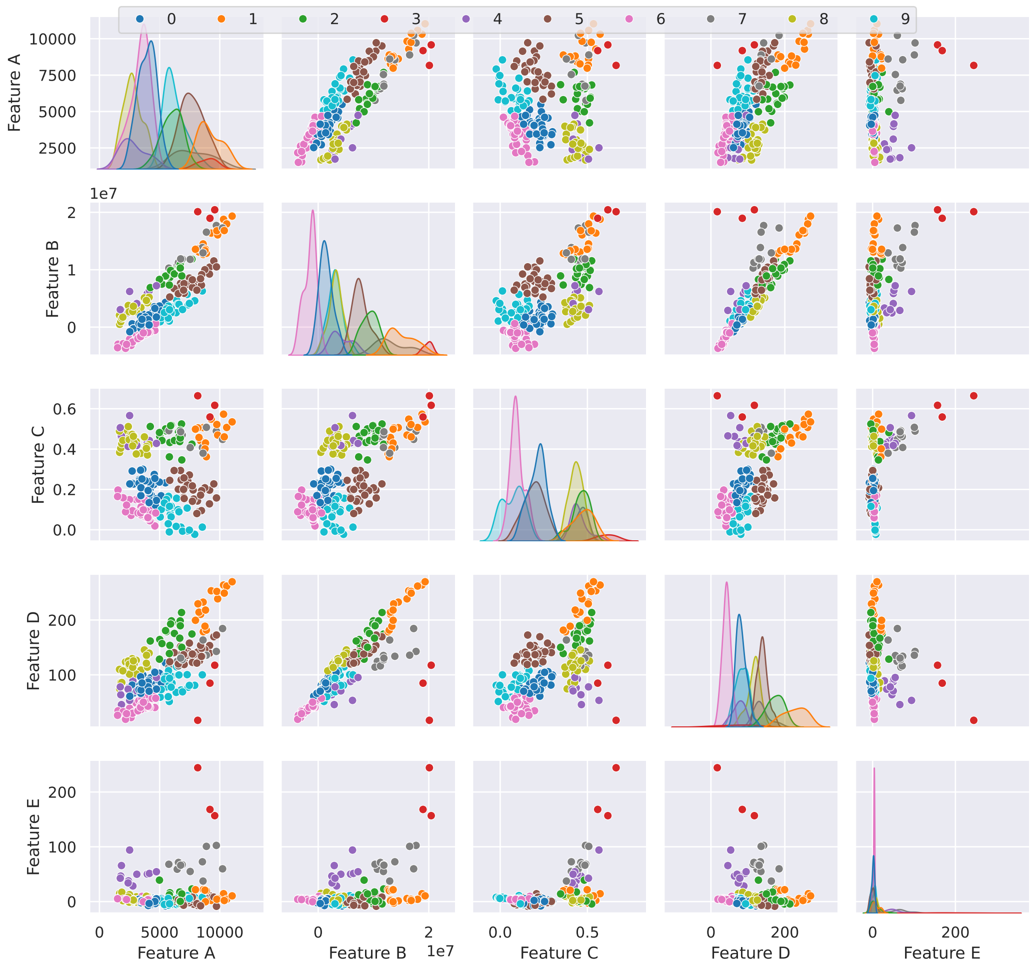


Figure 9 Clustering based on Euclidean distance in reduced space, with each of 10 clusters in different colors. Selecting an instance from each cluster gives us a good chance of obtaining a diverse set that roughly represents the instance space.

However, combining instance feature space analysis with instance generation could further enhance benchmark creation, though our current instance feature spaces turned out to be sufficiently dense to make this unnecessary.

Note that the MIPLIB 2017 benchmark suite [40] has been using a similar technique to select instances from the different submissions. As is natural for a benchmark suite for Mixed Integer Programming, they use a MIP to maximize the diversity of the selected instances. The International Timetabling Competition (ITC) 2019 [55] has explicitly used instance space analysis to generate diverse and representative benchmark instances, but their instances are easier to characterize than our geometric instances.

6 Conclusion

This paper detailed the development and hosting of seven CG:SHOP Challenges. We built a robust and scalable infrastructure using frameworks like Django, Docker, and Slurm, enabling seamless adaptations for each new challenge while maintaining consistency and reliability. Organizing these competitions involves significant efforts in problem selection, system development, instance generation, and participant support. Our data-driven instance selection approach effectively created diverse and representative benchmark problems, catering to a broad range of competitors, including students.

A key lesson learned regarding the competition infrastructure is to avoid overengineering and to instead prioritize simplicity and flexibility. Implementing straightforward monitoring and alerting systems proved more valuable than attempting to anticipate every possible issue, as the latter approach not only demands significant effort but also introduces complexity that can lead to additional problems, especially given the inevitability of unforeseen issues.

Another important lesson from individual competitions is to avoid overthinking and accept the presence of too many unknowns. As long as tie-breaker instances are included, the competition remains fair and the rankings meaningful. While it may be suboptimal if outcomes are determined by minute score differences, attempting to eliminate such occurrences is neither feasible nor worthwhile. Instead, effective communication and predictability are paramount.

To facilitate student participation, it is crucial to select problems that allow entry with simple ideas. This can be challenging in computational geometry, where geometric problems often require substantial background knowledge. However, exclusively choosing primarily combinatorial problems to avoid such challenges is not viable. Therefore, it is important to explore different directions each year and announce problems early, enabling instructors to integrate competitions into their curriculum and adequately prepare their students.

Looking ahead, we are committed to further developing the CG:SHOP Challenges and enhancing the competition experience in future iterations. We are eager to continue our work to support the computational geometry community and foster ongoing research and collaboration.

References

1 Bootstrap. `https://getbootstrap.com/`. Accessed: 2025-08-15.

2 Celebrate Google's coding competitions with a final round of programming fun. `http://g.co/hashcode`. Accessed: 2025-08-15.

3 Challenge ROADEF/EURO 2022: Trucks loading problem! `https://roadef.org/challenge`. Accessed: 2025-08-15.

4 Conan. `https://conan.io/`. Accessed: 2025-08-15.

5 DIMACS implementation challenges. `http://dimacs.rutgers.edu/programs/challenge/`. Accessed: 2025-08-15.

6 DISPLIB. `https://displib.github.io/`. Accessed: 2025-08-15.

7 Django. `https://www.djangoproject.com/`. Accessed: 2025-08-15.

8 Docker. `https://www.docker.com/`. Accessed: 2025-08-15.

9 Kaggle christmas challenge. `https://www.kaggle.com/c/christmas-challenge/data`. Accessed: 2025-08-15.

10 League of robot runners. `https://www.leagueofrobotrunners.org/`. Accessed: 2025-08-18.

11 Matplotlib. `https://matplotlib.org/`. Accessed: 2025-08-15.

12 MiniZinc - Challenge. `https://www.minizinc.org/challenge/`. Accessed: 2025-08-15.

13 Nagios. `https://www.nagios.org/`. Accessed: 2025-08-15.

14 Nginx. `https://nginx.org/`. Accessed: 2025-08-15.

15 Optimize the future of delivery: DH's VRPPD challenge. `https://co-at-work.zib.de/#challenge`. Accessed: 2025-08-15.

16 pandas. `https://pandas.pydata.org/`. Accessed: 2025-08-15.

17 Postgresql. `https://www.postgresql.org/`. Accessed: 2025-08-15.

18 Project jupyter. `https://jupyter.org/`. Accessed: 2025-08-15.

19 Python. `https://www.python.org/`. Accessed: 2025-08-15.

20 SAT competition 2025. `https://satcompetition.github.io/2025/`. Accessed: 2025-08-15.

21 scikit-build. `https://scikit-build.readthedocs.io/`. Accessed: 2025-08-15.

22 scikit-learn. `https://scikit-learn.org/`. Accessed: 2025-08-15.

23 seaborn. `https://seaborn.pydata.org/`. Accessed: 2025-08-15.

24 skbuild-conan. `https://github.com/d-krupke/skbuild-conan`. Accessed: 2025-08-15.

25 Slurm. `https://slurm.schedmd.com/`. Accessed: 2025-08-15.

26 Slurminade. `https://github.com/d-krupke/slurminade`. Accessed: 2025-08-15.

27 Ubuntu Linux LTS. `https://ubuntu.com/`. Accessed: 2025-08-15.

28 Verolog - All roads lead to verolog. `https://www.verolog.eu/`. Accessed: 2025-08-15.

29 Sina Abdipoor, Razali Yaakob, Say Leng Goh, Salwani Abdullah, Khairul Azhar Kasmiran, and Hazlina Hamdan. International timetabling competition 2019: A systematic literature review. In *2023 IEEE International Conference on Automatic Control and Intelligent Systems (I2CACIS)*, pages 22–27, 2023. `doi:10.1109/I2CACIS57635.2023.10193383`.

30 David Applegate, Robert Bixby, Vasek Chvatal, and William Cook. Concorde TSP solver. URL: `http://www.math.uwaterloo.ca/tsp/concorde.html`.

31 Max Bannach and Sebastian Berndt. PACE Solver Description: The PACE 2023 Parameterized Algorithms and Computational Experiments Challenge: Twinwidth. In Neeldhara Misra and Magnus Wahlström, editors, *18th International Symposium on Parameterized and Exact Computation (IPEC 2023)*, volume 285 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 35:1–35:14, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.IPEC.2023.35`.

32 Jeremias Berg and Jakob Nordström, editors. *LIPIcs, Volume 341, SAT 2025, Complete Volume*, volume 341 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.SAT.2025`.

33 Timo Berthold. Measuring the impact of primal heuristics. *Operations Research Letters*, 41(6):611–614, 2013. URL: `https://www.sciencedirect.com/science/article/pii/S0167637713001181`, `doi:10.1016/j.orl.2013.08.007`.

34 Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Joseph S. B. Mitchell. Computing convex partitions for point sets in the plane: The CG:SHOP Challenge 2020, 2020. `arXiv:2004.04207`.

35 Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Joseph S. B. Mitchell. Area-optimal simple polygonalizations: The CG Challenge 2019. *Journal of Experimental Algorithms*, 27:1–12, 2022. URL: `https://dl.acm.org/doi/10.1145/3504000`.

36 Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Joseph S. B. Mitchell. Computing coordinated motion plans for robot swarms: The CG:SHOP Challenge 2021. *Journal of Experimental Algorithms*, 27:3.1:1–3.1:12, 2022. URL: `https://dl.acm.org/doi/10.1145/3532773`.

37 Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Stefan Schirra. Minimum partition into plane subgraphs: The CG:SHOP Challenge 2022. *Journal of Experimental Algorithms*, 28:1.9:1–1.9:13, 2022. URL: `https://dl.acm.org/doi/10.1145/3604907`.

38 Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Stefan Schirra. Maximum polygon packing: The CG:SHOP Challenge 2024, 2024. `arXiv:2403.16203`.

39 Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Stefan Schirra. Minimum coverage by convex polygons: The CG:SHOP Challenge 2023. *Computing in Geometry and Topology*, 5(4):2:1–2:12, 2026. This issue. URL: `https://doi.org/10.57717/cgt.v5i4.115`.

40 Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, et al. MIPLIB 2017: data-driven compilation of the 6th mixed-integer programming library. *Mathematical Programming Computation*, 13(3):443–490, 2021.

41 Ernestine Großmann, Tobias Heuer, Christian Schulz, and Darren Strash. The PACE 2022 Parameterized Algorithms and Computational Experiments Challenge: Directed Feedback Vertex Set. In Holger Dell and Jesper Nederlof, editors, *17th International Symposium on Parameterized and Exact Computation (IPEC 2022)*, volume 249 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:18, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.IPEC.2022.26`.

42 Andreas Haas. Solving large-scale minimum-weight triangulation instances to provable optimality. In Bettina Speckmann and Csaba D. Tóth, editors, *34th International Symposium on Computational Geometry, SoCG 2018, June 11-14, 2018, Budapest, Hungary*, volume 99 of *LIPIcs*, pages 44:1–44:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. `doi:10.4230/LIPIcs.SoCG.2018.44`.

43 Wenzel Jakob, Jason Rhinelander, and Dean Moldovan. pybind11–seamless operability between C++ 11 and python. *URL: https://github.com/pybind/pybind11*, 2024.

44 Serdar Kadıoğlu and Bernard Kleynhans. The design and organization of educational competitions with anonymous and real-time leaderboards in academic and industrial settings. *arXiv preprint arXiv:2402.07936*, 2024.

45 John R. Rice. The algorithm selection problem. In *Advances in computers*, volume 15, pages 65–118. Elsevier, 1976.

46 Paul Shaw, editor. *30th International Conference on Principles and Practice of Constraint Programming, CP 2024, September 2-6, 2024, Girona, Spain*, volume 307 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

47 Kate Smith-Miles, Davaatseren Baatar, Brendan Wreford, and Rhyd Lewis. Towards objective measures of algorithm performance across instance space. *Comput. Oper. Res.*, 45:12–24, 2014. `doi:10.1016/j.cor.2013.11.015`.

48 Kate Smith-Miles and Simon Bowly. Generating new test instances by evolving in instance space. *Comput. Oper. Res.*, 63:102–113, 2015. `doi:10.1016/j.cor.2015.04.022`.

49 Kate Smith-Miles and Leo Lopes. Measuring instance difficulty for combinatorial optimization problems. *Comput. Oper. Res.*, 39(5):875–889, 2012. `doi:10.1016/j.cor.2011.07.006`.

50 Kate Smith-Miles and Thomas T. Tan. Measuring algorithm footprints in instance space. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2012, Brisbane, Australia, June 10-15, 2012*, pages 1–8. IEEE, 2012. `doi:10.1109/CEC.2012.6252992`.

51 Kate Smith-Miles and Jano I. van Hemert. Discovering the suitability of optimisation algorithms by learning from evolved instances. *Ann. Math. Artif. Intell.*, 61(2):87–104, 2011. `doi:10.1007/s10472-011-9230-5`.

52 Kate Smith-Miles, Jano I. van Hemert, and Xin Yu Lim. Understanding TSP difficulty by learning from evolved instances. In Christian Blum and Roberto Battiti, editors, *Learning and*

Intelligent Optimization, 4th International Conference, LION 4, Venice, Italy, January 18-22, 2010. Selected Papers, volume 6073 of *Lecture Notes in Computer Science*, pages 266–280. Springer, 2010. doi:10.1007/978-3-642-13800-3_29.

53 Kate Amanda Smith-Miles. Towards insightful algorithm selection for optimisation using meta-learning concepts. In *Proceedings of the International Joint Conference on Neural Networks, IJCNN 2008, part of the IEEE World Congress on Computational Intelligence, WCCI 2008, Hong Kong, China, June 1-6, 2008*, pages 4118–4124. IEEE, 2008. doi:10.1109/IJCNN.2008.4634391.

54 The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 6.0.1 edition, 2024. URL: https://doc.cgal.org/6.0.1/Manual/packages.html.

55 David Van Bulck and Dries Goossens. The international timetabling competition on sports timetabling (itc2021). *European Journal of Operational Research*, 308(3):1249–1267, 2023.

56 Szymon Wasik, Maciej Antczak, Jan Badura, Artur Laskowski, and Tomasz Sternal. A survey on online judge systems and their applications. *ACM Computing Surveys (CSUR)*, 51(1):1–34, 2018.