University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Krzysztof Małysa
Student no. 394442

Multi-process sandbox for unprivileged users on Linux

Master's thesis
in COMPUTER SCIENCE

Supervisor:
dr Janina Mincer-Daszkiewicz
Institute of Informatics

Warsaw, December 2023

Abstract

TODO

Keywords

sandboxing, security, Linux, secure execution, arbitrary code execution, judging system,

Thesis domain (Socrates-Erasmus subject area codes)

11.3 Informatics, Computer Science

Subject classification

Security and privacy – Systems security – Operating systems security

Tytuł pracy w języku polskim

Wieloprocesowa piaskownica dla nieuprzywilejowanych użytkowników systemu Linux

Contents

Chapter 1

Introduction

1.1. Background

Secure execution environments are commonplace these days, from containers and virtual machines on servers to sandboxes on laptop and smartphones — most of which run on Linux. They are used to securely execute untrusted code, as well as trusted programs to prevent damage escalation in the event of unknown vulnerabilities. Their key features are isolation, limiting resource usage, and accounting for resource consumption.

The features of Linux allow the creation of simple yet effective and efficient secure environments. They work at application runtime, so in most cases existing software does not need to be adapted to use them. This makes them easily applicable, and explains why their adoption is growing.

In this thesis, the most important application of sandboxing are online judge systems. Online judge systems have beneficial role in programming education and competitive programming. They allow testing user-provided solution to a specific problem. The solution is run on a predefined test cases in order to check if it is valid. In such platforms isolating the compilation and running of the tested program is essential to provide security and robustness of the platform itself.

Historically, isolation techniques evolved together with the online judge platforms. The most primitive (yet insecure) was usage of `chroot(2)` [29] to restrict access to part of the filesystem. To increase isolation virtual machines were used [2]. Later, containerization became a new way to provide isolation [20, 40].

Online education platforms greatly facilitate teaching and learning programming. They provide quick feedback on the correctness of the code the user submits. They are used in schools and universities and provide great learning opportunities for all.

Moreover, a versatile sandbox has applications outside online judging platforms. For example, it can be used to sanitize compiling a PDF form LaTeX sources or for safe execution of untrusted server-side scripts in web applications.

1.2. Goal of the thesis

The goal of the thesis is to design, implement and integrate a new sandbox for the Sim project [16]. The Sim project is an online platform for preparing people for and carrying out algorithmic contests. The project started in 2012 and is developed by me since the beginning. It is used at the XIII High School in Szczecin and programming camps to teach young people programming and algorithms. It has an online judge with a sandbox specially developed for this use case. Over the years the sandbox became a limitation. It only allows running a single-threaded statically linked executable of programs written in C, C++ or Pascal. The new sandbox will allow supporting more programming languages and improve security of the tested program compilation stage.

1.2.1. Requirements

The new sandbox needs to be optimized for running short-running programs as well as have minimal runtime overhead. Most of the test cases the tested program is run on are small and it completes them in less than 10ms. The goal is to allow hundreds of such sort-running runs per second, hence optimizing for short-running programs is important. However, minimizing overhead of the sandbox during the run is also important i.e. if the program runs X ms normally, the objective is that the program inside the new sandbox will also run approximately X ms.

The new sandbox needs to be versatile. It will be used to secure the compilation of the tested programs as well as running of the tested programs. Compilation is a complicated process that involves parsing, translating, optimizing and linking the final program. For languages like C, C++ and Rust it involves running several executables in coordination e.g. compiler and linker i.e. more than one process at a time — the sandbox needs to support that.

Sandboxing needs to have a low overhead. Apart from small test where a tested program runs quickly (a matter of milliseconds), almost always the tested program is run on big test cases, where it may need several seconds for it to solve the problem. Increasing this time as little as possible while the tested program is running inside the sandbox is one of the primary objectives.

It often requires running several executables e.g. compiler and linker, so allowing a single process inside sandbox is not enough. Sandboxing the tested program is simpler, because it is a single process. But since it is often short-running, the overhead needs to be minimal.

The sandbox needs to allow limiting resources. Real time, CPU time, memory – these need to be limited not only for the robustness of the platform, but specific problems require different limits. The goal of some problems is to solve it with very restricted memory e.g. find a missing integer in a random permutation of integers $1, \ldots, n$ without one element, but in $O(1)$ memory.

The sandbox needs to account resource usage. For every test, the user is presented with consumed memory and CPU time by their solution. The sandbox needs to provide this information.

The last requirement is the sandbox will not require any privileges. There is a tool called Sip [17] for preparing the problem packages for the Sim platform. One of the purposes of the tool is to run the solutions inside the same secure environment as on the Sim platform. The user should not need any privileges to run this tool, so the sandbox should not require them either.

1.2.2. Existing solutions

Approaches to form a secure execution environments differ. One of them is virtualization or emulation e.g. QEMU [27] and KVM [26], VirtualBox [28], VMWare Workstation [44]. Although powerful and effective, they come with an enormous overhead i.e. booting up an entire operating system. Moreover, emulation noticeably slows down the runtime of an emulated application, rendering such solutions inapplicable.

Containers provide much lower overhead: setup of an order of milliseconds and negligible runtime overhead. But, Docker [22], LXC [1] require root privileges to create a container. systemd-nspawn [42] requires root privileges to run.

Rootless containers are containers [38] that can be created and run by an unprivileged user are the almost perfect solution to the problem. They provide almost all of the functionality of the normal containers but without the need to engage a privileged user. However, they often use setuid binaries and that is undesirable [39]. Also they are not optimized to run sequences of short-running programs. In this thesis we will create a sandbox that uses the same techniques as rootless containers but will be optimized for running sequences of short-running programs.

1.3. Structure of the Thesis

Chapter 2 contains overview of sandboxing techniques and existing implementations and comparative analysis of them. Details of design and architecture are described in Chapter 3. Implementation is described in Chapter 4. Chapter 5 contains performance evaluation of the final implementation and impact of some optimizations. Finally, Chapter 6 contains conclusions and suggestions for further work.

Chapter 2

Problem overview

2.1. Overview of Sandboxing Techniques

During the first programming competitions, the human judges manually read and verified the source code of the contestants' solutions [43]. Over time this became infeasible and gave birth to automatic judge systems.

To prevent people from interfering with the normal workflow of the competition, e.g. Denial of Service Attack by exhausting memory resources, the automatic judge systems need a secure way to compile and execute a contest's solution. This is where sandboxes come into place.

First sandboxes required modification of the OS kernel [36, 7, 8, 10, 13]. While they had little run-time overhead, some of them were limited to single-threaded applications [24].

Later, as support for process tracing matured, `ptrace`-based sandboxes arose [19, 12, 11]. The problem with those solutions is the overhead that varies from around 75% [23] to 160% [20] for syscall-intensive programs. This overhead however, does not affect programming contest fairness much [18]. Supporting multi-threaded and multi-process programs while using `ptrace` is tricky, but possible [11], because of Time of Check/Time of Use (TOCTOU) problem [5]. `ptrace`-based sandbox needs to inspect syscall arguments. To do so it has to read them, but the multi-threaded or multi-process program can change the indirect argument after the reading but before the kernel uses the argument. This creates a dangerous race condition that has to be addressed.

Finally, after the kernel support for containerization materialized, namespace and cgroup based sandboxes came into place [20, 25, 37, 9, 6, 40]. Contrary to ptrace-based sandboxes, namespace-based sandboxes have negligible runtime overhead [20]. Moreover, they don't require modifications of the Linux kernel and work on major Linux distributions out of the box.

2.2. Existing Implementations

2.2.1. Modifying OS kernel

Systrace

Systrace [36] intercepts all system calls in the kernel. It then decides if the syscall is safe by first checking a static list of safe syscalls. This step exists to reduce sandboxing performance overhead. If the syscall is not on the list, Systrace consults user space for a decision.

The system avoids TOCTOU problem [5] by copying syscall arguments to kernel memory before asking user space for a decision.

Janus

Janus [7] adds a module to extend Linux `ptrace` API. Policies are defined using configuration files. By default all syscalls are denied. The configuration directive refers to the policy module that provides the logic for deciding whether to allow a particular system call or not. For example, `path` module could be used to restrict IO on certain file paths.

Ostia

Ostia [8] instead of filtering system calls delegates them to an external agent that performs syscalls on behalf of the sandboxed process. Authors emphasize that such architecture simplifies the system and protects from TOCTOU problems [5].

Ostia is implemented as two components: a small kernel module and a user space part. The module intercepts the syscall and copies its arguments via IPC link to the user space agent. The agent decides whether the call should be allowed, executes it and returns the results back over the IPC link. Worth noting is that not all syscalls have to be delegated — some can be always allowed while others always denied.

TxBox

TxBox [10] introduces system-level transaction support. Impact of the untrusted insecure code is limited by rolling back the system state after the execution. This provides strong isolation and works with arbitrary executables but requires significant out-of-tree patches of the OS kernel.

MiniBox

MiniBox [13] is a two-way sandbox that protects operating system from the application as well as application from the operating system. A modified version of TrustVisor [21] hypervisor runs OS and sandboxed application separately in a Mutually Isolated Execution Environment. The hypervisor is the only communication channel between the isolated application and the regular OS. This way application is protected from the malicious operating system. To protect the OS from the application, MiniBox uses Software Fault Isolation techniques from NaCl [45].

SACO sandbox

South African Computer Olympiad (SACO) sandbox [24] inserts a custom kernel module that hooks up to Linux Security Module infrastructure. Although it has negligible time and memory overhead, it only supports single-threaded programs.

2.2.2. `ptrace`-based

MO sandbox

MO sandbox [19, 12] allows only single-threaded programs. It simply inspects arguments using `ptrace` and uses `setrlimit` [31] to limit resources. It is used by USA Computer Olympiad (USACO).

MBOX

MBOX [11] requires no superuser privileges. It makes use of seccomp BPF system call filtering to restrict allowed syscalls. BPF filtering is effective only for non-indirect arguments. To address this issue, the installed BPF filter notifies the `ptrace` monitoring process if further argument inspection is necessary to make a decision. To avoid TOCTOU problem [5], the MBOX allocates a read-only page to which it copies the indirect arguments before inspecting them and rewrites the syscall to use the rewritten arguments. The copied arguments are protected against modification because changing page access permissions is impossible without a syscall.

2.2.3. Using Linux namespaces

Firejail

Firejail [25] uses seccomp BPF system call filtering and mount namespaces to restrict filesystem access. Similarly it uses process namespaces to limit view of running processes and network namespaces to restrict access to network devices. However, Firejail uses a `setuid` [34] helper binary to achieve that. It allows resource limiting through `prlimit` [31].

nsjail

nsjail [9] uses Linux namespaces, seccomp BPF system call filtering, `setrlimit` [31] and cgroups to limit resources. It does not require superuser privileges. However, it is not optimized for running short-running programs. Also, it does not provide statistics of the run.

nsroot

nsroot [37] does not support resource limiting. It only makes use of Linux namespaces to restrict view of the file system, IPC and network devices.

Flatpak

Flatpak [6], previously xdg-app, is a software packaging and sandboxing tool. Internally, it uses Bubblewrap sandbox. The Bubblewrap [14] is a `setuid` [34] program that uses Linux namespaces and seccomp filters.

New Contest Sandbox

New Contest Sandbox [20] uses Linux namespaces and cgroups but not seccomp filters. It is used by Moe modular contest system (2012) [20]. Linux namespaces and cgroups have negligible overhead compared to `ptrace`.

APAC

APAC (Automatic Programming Assignment Checker) [40] uses Docker for sandboxing. It sets up a container for each run. Docker uses runC under the hood. While runtime overhead of Docker is low, the setup phase is primary source of overhead for short-running programs.

runC

runC [3] uses the same features of Linux kernel as nsjail. However, configuration is stored as files instead of passed as command-line arguments. It has a special `rootless` mode which does not require superuser privileges. Given all of the above however, it is not optimized for running short-running programs. runC is used internally by Docker.

2.2.4. Other

Google Native Client

Native Client (NaCl) [45] uses static analysis and Software Fault Isolation. After the static analysis, the program runs at native speed but requires recompilation with special compiler and libraries. NaCl only works for x86 architecture.

2.3. Conclusion

Many sandboxing solutions exist. From all of the above, closest to our requirements is nsjail (see Section 2.2.3). However, it is not optimised for running short-running programs. In fact, none of the above solutions is optimised to run hundreds of short-running programs per second.

Considering the similarities of nsjail and our solution, in the performance analysis we will compare our sandbox to nsjail sandbox.

Chapter 3

Design and Architecture

3.1. Client-server architecture

From the start the sandbox was based on the client-server architecture. This was the choice to minimize process cloning overhead [15], since it is a costly operation and happens for every sandboxing request. fork/clone needs to clone the whole address space of the process and the client process could have a large address space. Server process that is executed from a separate executable has a minimal required address space size therefore the cloning overhead is minimal for every request. Moreover, this architecture easily and safely allows sharing as much work as possible between the sandboxing requests which is the key to low overhead of running short-running programs.

The client spawns the sandboxing server and sends sandboxing requests via UNIX domain socket to the server. This is illustrated on Figure 3.1. The request contains executable, arguments, namespace configuration, resource limits, seccomp BPF filters and a pipe through which the result will be sent back.

At startup, the server creates cgroup hierarchy and some namespaces so that they won't have to be created later or their creation will be faster. Other utilities are also setup here to do it once instead of for every request. Then it starts accepting requests.

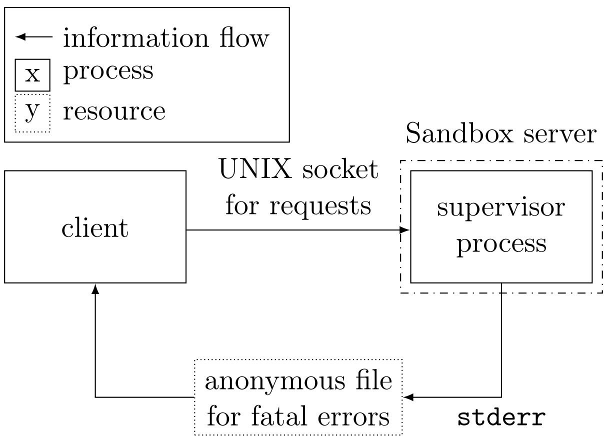

Figure 3.1: Sandbox server waits for requests. Client sends requests through the UNIX socket. Sandbox server will die on fatal error leaving the error message for the client in the anonymous file.

3.1.1. Sandboxing request handling

For each request, the server process (aka supervisor) spawns the PID 1 process of the new PID namespace. Then the init process setups namespaces and some of the resource limits. Finally the PID 1 process spawns the tracee process that finishes configuration and executes the requested executable. So for each sandboxing request we spawn exactly 2 processes. However, the executed program can spawn new processes — each of them is referred to as a "tracee process". The PID 1 process is necessary for a couple of reasons:

- It reaps the zombie processes in the tracee PID namespace.
- It allows locking mount-points in the mount namespace. The tracee process is spawned in a new user and mount namespace. Mounts are performed by the PID 1 process, therefore all mounts become locked together and cannot be individually unmounted by the tracee [32]. These mounts cannot be performed by the supervisor process instead, because it would alter the mount namespace for subsequent requests.
- Inside a PID namespace, sending signals to the PID 1 process is allowed only for signals that the PID 1 process installed signal handler for. This could change the behavior for some programs, therefore a helper PID 1 process is needed.

This is shown on Figure 3.2.

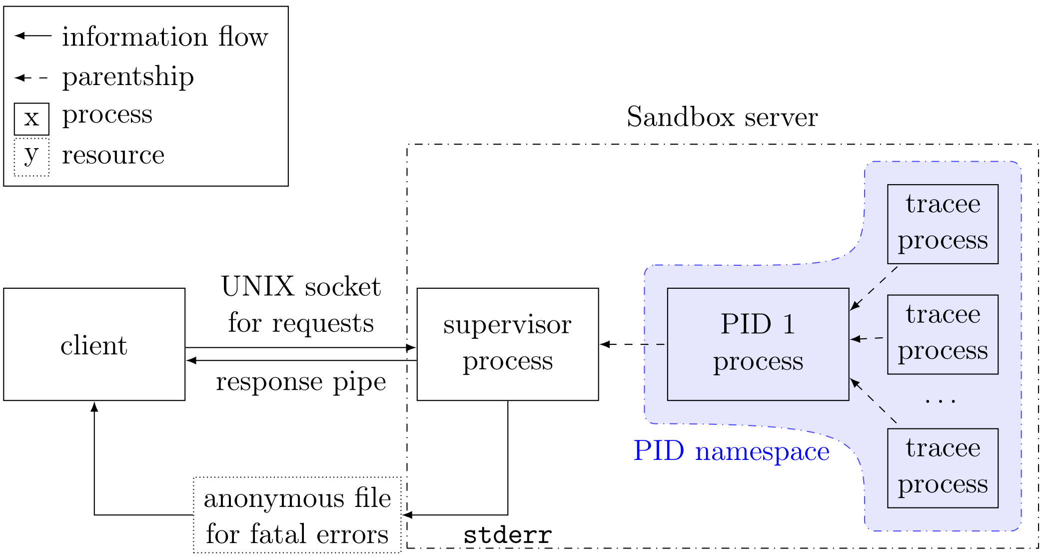

Figure 3.2: Sandbox server handles a request, at the moment after executing the requested executable. Sandbox server will die on fatal error leaving the error message for the client in the anonymous file. Sandbox server consist of the supervisor process and its child — the PID 1 process that is spawned for each request. The PID 1 process performs a role of the init process in the PID namespace of the tracee processes.

3.2. Cgroups

The server gains write access to cgroup hierarchy by being executed through `systemd-run --user --scope --property=Delegate=yes --collect`. It enables `pid`, `memory`, and `cpu` controllers for the below subgroups.

At startup, the server process creates the cgroup v2 hierarchy that looks as follows:

- `/supervisor` — cgroup of the supervisor process,
- `/pid1` — cgroup of the PID 1 process,
- `/tracee` — cgroup of the tracee processes.

After creation of the hierarchy it places the supervisor process in its cgroup. Subsequent processes are placed in their cgroups by making use of `CLONE_INTO_CGROUP` flag.
`/tracee` cgroup allows:

- Killing all tracee processes by writing 1 to `/tracee/cgroup.kill` file.
- Reading CPU user and CPU system time via `/tracee/cpu.stat` file.
- Reading peak memory usage via `/tracee/memory.peak` file.
- Setting process/tread number limit by writing `/tracee/pids.max`.
- Setting memory hard limit by writing `/tracee/memory.max`.
- Setting CPU usage limit by writing `/tracee/cpu.max`.
- Disabling PSI accounting to reduce the sandboxing overhead by writing 0 to `/tracee/cgroup.pressure` file.

`/tracee` cgroup needs to be deleted and recreated after each request to reset `/tracee/cpu.stat` and `/tracee/memory.peak` files.

3.3. Linux namespaces

Linux allows unprivileged users to create user namespaces only. However, after entering a new user namespace the process gains all privileges inside the namespace and can create other namespaces.
The supervisor process creates the following namespaces:

- user namespace — in order to create other namespaces and hide user ID and group ID,
- mount namespace — to allow mounting detached cgroups v2 hierarchy,
- cgroup namespace — to allow mounting detached cgroups v2 hierarchy,
- network namespace — to disconnect every tracee from network devices, done once, as it is costly,
- IPC namespace — to isolate every tracee from other processes' IPC, done once, for optimization,
- UTS namespace — to isolate every tracee from host's hostname, done once, for optimization,
- time namespace — to isolate every tracee from host's time namespace, done once, for optimization.

The PID 1 process creates the following namespaces:

- user namespace — in order to create other namespaces and hide user ID and group ID,
- mount namespace — to allow mounting requested mount-point hierarchy,
- PID namespace — to isolate tracee from accessing other processes.

The tracee process creates the following namespaces:

- user namespace — in order to create other namespaces and hide user ID and group ID and lock the mount tree,
- mount namespace — in order to lock the mount tree created by PID 1 process.

The listed namespaces hierarchy is illustrated on Figure 3.3.

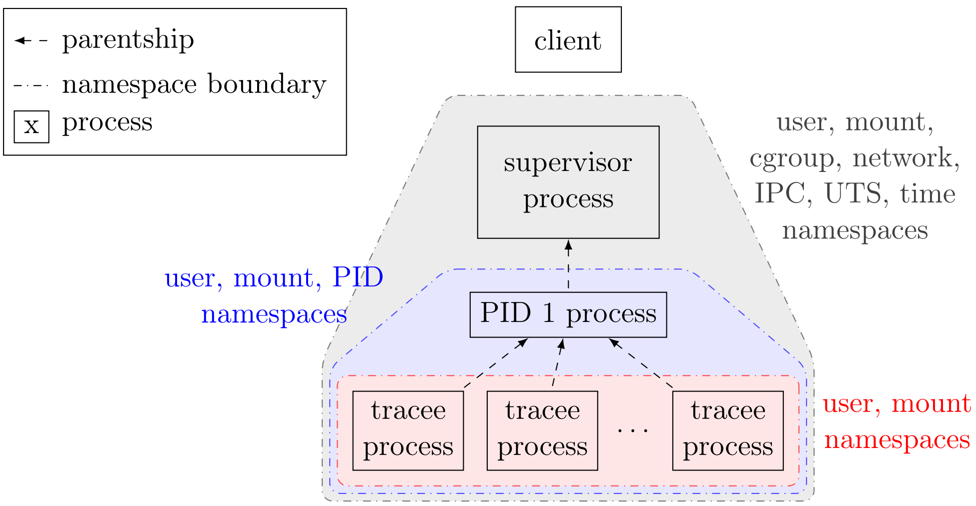

Figure 3.3: Namespaces hierarchy of the sandbox server processes.

3.4. Inter-process communication

The client sends requests via UNIX domain socket to the supervisor process. The results are sent via a pipe attached to the request. The pipe is attached to the request as a file descriptor using `SCM_RIGHTS` control message [35].

The supervisor, the PID 1 and the tracee processes communicate via shared anonymous memory page. Figure 3.4 illustrates this communication. Such communication requires no syscalls, is fast and reliable. This page is automatically unmapped upon `execveat` syscall [30] in the tracee process, so it is protected from the tracee access.

3.5. Capabilities

The supervisor process drops all capabilities, sets securebits and `NO_NEW_PRIVS` flag. This ensures minimal possible capabilities in its and ancestor user namespace and prevents gaining any new privileges.

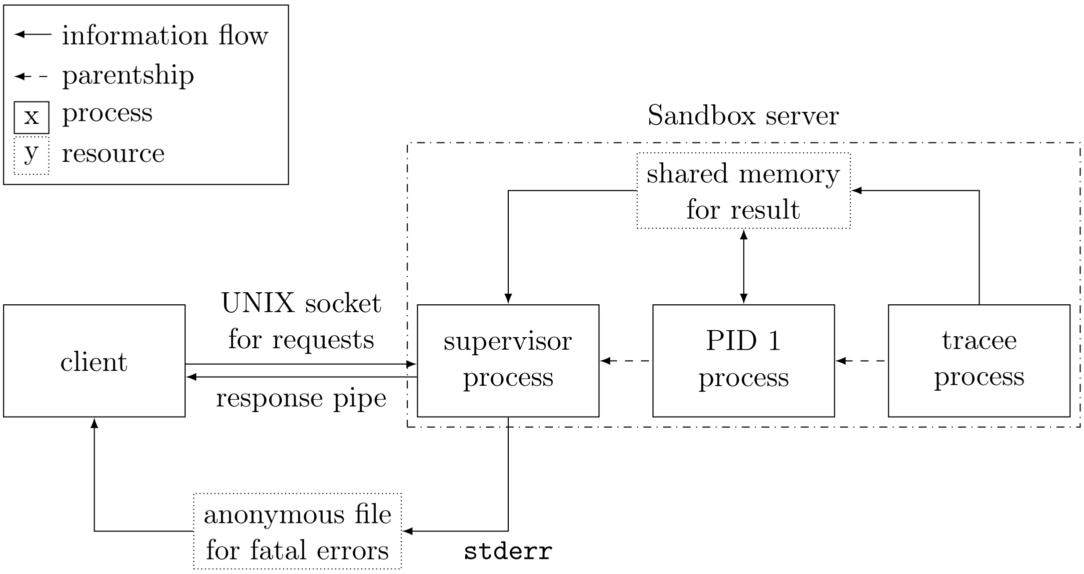

Figure 3.4: Sandbox server handles a request, at the moment before executing the requested executable. Sandbox server will die on fatal error leaving the error message for the client in the anonymous file. Sandbox server consist of the supervisor process and its child — the PID 1 process that is spawned for each request and its child — the tracee process that will execute the requested executable. The tracee process saves in the shared memory the time just before `execveat` and signals the PID 1 process. The PID 1 process reads the time saved in the shared memory and starts the real and CPU timers. After the tracee process dies, the PID 1 process writes exit code and status of the tracee process and the time it died. Moreover, the shared memory is used to communicate fatal errors to the supervisor process.

3.6. Hardening

The PID 1 process, after spawning the tracee enters a new cgroup namespace to limit view of other namespaces i.e. if the tracee somehow takes control of the PID 1 process, it will not be able to raise its PID and memory limit. Moreover a seccomp filter is installed to limit allowed syscalls only to those needed for reaping the orphaned zombie process, managing time limits and exiting upon tracee death.

3.7. Conclusion

Client-server architecture allows time-performance optimizations. Furthermore, it allows more common work to be done once and simplifies implementation. For instance, file descriptors do not leak to other processes because there are no threads that could fork a new process. Request handling requires creation of 2 processes — the PID 1 process and the tracee process that later executes the requested program. Resource limits and accounting is mostly performed by cgroups. Isolation is achieved by deft usage of Linux namespaces.

Chapter 4

Implementation

The project is written in C++, as it is a low-overhead, low-level language, but more convenient than C. Git is used as a Version Control System to track incremental implementation. Invaluable tool used during development was `strace` [41]. It allowed easy inspection of system calls and their return values without any modification to the source code.

4.1. Interface

The client has to spawn the sandbox server — the supervisor process. It then operates on the connection handle. Using the connection handle, the client can send requests to execute programs in the sandbox.

After sending a request, a request handle object is constructed. It can be used to obtain result of the execution, cancel execution or kill the tracee processes. Canceling execution is useful in case of errors in the client, where the result of the execution as well as the execution itself are no longer needed. Upon cancellation of the request, the tracee processes are immediately killed. The result of running the request is discarded. Canceling an already finished request discards the result. Canceling the request before the server started handling it causes it to be skipped. Killing an already finished request is no-op. Killing the request before the server started handling it causes tracee to be immediately killed after the `execveat`.

The client can then await the result of the request using the request handle. The result can either be a successful result or an error with a textual description. This error is not fatal to the supervisor process i.e. new requests can be sent. The successful result consists of the exit code and status, and runtime statistics:

- real time,
- CPU user and CPU system time,
- peak tracee cgroup memory usage.

Each request has a set of accompanying options:

- Optional `stdin`, `stdout`, and `stderr` file descriptors. If optional is specified as empty, `/dev/null` is opened as the file descriptor.
- Environment as an array view of string views.
- Linux namespace configuration:

  - user ID and group ID mapping,
  - mounts and new root mount,

- Cgroup resource limits: process and thread limit, memory limit, CPU maximum bandwidth.

- `prlimit` hard limits.

- Real time limit.

- CPU time limit.

- Seccomp BPF filter as a file descriptor. The decision to pass it as a file descriptor is that it lowers the overhead of repeatedly using the same filter — a common scenario in a judge system. Only the file descriptor needs to be sent with each request instead of the whole BPF filter content. This allows the filter to be compiled once and passed for multiple requests with minimal overhead. An alternative is to extend the API to save seccomp filters but it was considered unnecessary given how small is the overhead of passing a single file descriptor.

4.2. Time limits

The PID 1 process controls the time limits. The tracee process, just before `execveat` saves current real time from `CLOCK_MONOTONIC_RAW` and CPU time from `cpu.stat` tracee cgroup file to the shared memory (see Figure 3.4). The problem with `cpu.stat` file is that it is updated infrequently. For a young tracee process, this file often reports consumed CPU time equal to 0 microseconds instead of a few hundred microseconds. Fortunately, executing `sched_yield()` system call forces recalculation of the file and the values are no longer 0. This is why this syscall is required as allowed in the seccomp BPF filter.

4.2.1. Real time limit

After saving the current real time the tracee process signals the PID 1 process with `SIGUSR2`. The PID 1 process reads the saved real time and sets up a POSIX timer to expire at the moment of saved time + real time limit. When the timer expires, `SIGUSR1` is sent by the kernel to the PID 1 process and it terminates all tracee processes by writing 1 to the `cgroup.kill` file of the tracee cgroup.

4.2.2. CPU time limit

In case the tracee is not restricted to one process, the setup is analogous to real time except that there is no CPU timer for a cgroup of processes. Instead we calculate minimal period of time in which the CPU time limit could expire as follows: $\frac{\text{remaining cpu time}}{\text{max parallelism}}$, where max parallelism equals: $\min(\text{available threads}, \texttt{process\_num\_limit}, \texttt{cpu\_max\_bandwidth}\text{ in threads})$. Upon the timer expiration the remaining cpu time is recalculated and the timer is rescheduled if the remaining cpu time is greater than 0. Timer expiration is signaled by the kernel as signal `SIGXCPU`. To prevent polling, the minimal timer expiration period is capped to have minimum value of 1ms — this gives at most 1000 checks per second.

In case the tracee is restricted to one process, the setup is different. After saving the current CPU time the tracee process signals the PID 1 process through a pipe. A signal cannot be used because `timer_create` syscall and `clock_getcpuclockid` library function are not marked async-signal-safe — they are not specified to be safe to call inside a signal handler. An `eventfd` cannot be used either, because if tracee dies before writing to the `eventfd`, the PID 1 process will wait indefinitely on the `read` syscall. With a pipe, `read` syscall returns 0 when the other end becomes closed. With the limit of one process we use the CPU timer of the tracee process and set up a timer to expire when the tracee exceeds the CPU time limit.

In both cases, when the CPU time limit is exceeded, the PID 1 process is signaled about it and it terminates all tracee processes by writing 1 to `cgroup.kill` file of the tracee cgroup.

4.3. Runtime statistics

After the main tracee process (the first spawned process) exits, the PID 1 process saves the current real time and the exit status in the shared memory, unless the tracee set an error, and exits. The kernel kills all remaining tracee processes (because the PID namespace's init process died). After the PID 1 process exits, the supervisor process reads the shared memory (see Figure 3.4). It checks if there is an error of either tracee or PID 1 process. If there is one, it becomes the result of the request. If there is none, the supervisor process calculates:

- real time using formula: time of tracee death − saved `execveat` real time,
- CPU time using formula: CPU time read from `cpu.stat` file−saved `execveat` CPU time,
- Peak memory usage by reading `memory.peak` tracee cgroup file.

4.4. Error handling

Errors in the supervisor process are considered fatal and are reported by writing to `stderr`. After writing errors, the supervisor process exits immediately. When the client tries to read the request result, the read will fail with `read` returning unexpected value 0. The client then ensures the supervisor process is dead (in case the communication failed) and tries to read the error the supervisor wrote. If the client finds one it throws exception with this error, otherwise it throws the exception with the `read` error.

The PID 1 process and the tracee process write error to the shared memory (see Figure 3.4) and exit immediately. The supervisor process reports these errors as a request result.

4.5. Request sending and receiving

The request is sent via UNIX domain socket (see Figure 3.1). The request consists of a constant-length header with file descriptors and a variable length body. The request header contains only the length of the request body. The request body contains all parameters of the request that are serialized to a custom binary format.

4.6. File descriptors

The sandbox server closes all file descriptors except the UNIX socket fd and opens `/dev/null` as `stdin`, `stdout`, and `stderr`. This is a small optimization, in case the request does not specify a standard file descriptor. For instance, if the request is to execute a program without `stdin`, the sandbox server has to set up the `stdin` of the tracee process to be `/dev/null` opened for reading. However, the tracee process inherits the file descriptors of the PID 1 process that inherits the file descriptors of the supervisor process. The supervisor process has already opened `/dev/null` as the `stdin` file descriptor. Therefore no action is needed in the PID 1 process and the tracee process for `stdin` of the tracee process to be `/dev/null` opened for reading. The same principle applies to `stdout` and `stderr` file descriptors.

All file descriptors are opened with `O_CLOEXEC` flag so that they will not leak to the executed process. A unit test to check if any file descriptor leaks to the sandboxed program is in the test suite.

The PID 1 process inherits all request standard file descriptors and passes them to the tracee process. It has to close them after spawning the tracee process. To see why, let's consider a pipe of which one end is passed as a `stdin` to the sandboxed program. A pipe is broken if all file descriptors of one end become closed. If the PID 1 process did not close the standard file descriptors of the tracee, the pipe could not become broken until the PID 1 process dies. This changes the semantics of the pipe if the program is run inside the sandbox and is undesirable. Moreover, for hardening purposes the PID 1 process closes all unnecessary file descriptors after spawning the tracee — in case, the tracee somehow takes control of the PID 1 process.

4.7. Canceling the request

The response to the request is passed via pipe that is provided alongside the request. If the pipe becomes broken i.e. the client closes the read end of the pipe, the request is considered cancelled and is immediately discarded if currently handled and omitted otherwise.

4.8. Killing the request

The `eventfd` file descriptor is sent with the request. The supervisor process monitors this file descriptor and if it becomes readable i.e. the client writes a value to it, the tracee is killed immediately. To avoid false-positive errors (the tracee process is killed unexpectedly), if the request is killed before `execveat` syscall executing the requested program, killing of the tracee is delayed to the `execve` call.

4.9. Sandbox server upon client death

The supervisor process monitors the UNIX socket through which the requests flow in. If the other end becomes read and write closed, the supervisor recognises it as a death of the client process and dies immediately. The PID 1 process is configured to die upon the supervisor process death, and the kernel kills all tracee processes when the PID 1 process dies. Therefore, all server processes die.

4.10. PID 1 process upon supervisor death

The PID 1 process configures the kernel to kill it upon the supervisor process death. This is done using `prctl`'s option `PR_SET_PDEATHSIG`. However, if the supervisor dies before the kernel configures the PID 1 process to die, the PID 1 process will still be alive and waste the resources. To solve this, one could check if the `getppid()` returns the expected PID of the supervisor process. However, this will not work, since the PID 1 process is in a new PID namespace, and `getppid()` will always return 0. A reliable solution is to pass a pidfd file descriptor of the supervisor process and check if the supervisor process is dead by checking if the pidfd file descriptor became readable. This way, upon supervisor process death, the PID 1 process is either killed by the kernel or it detects the death of the supervisor process and kills itself.

4.11. Signals

Signals are another way the processes can communicate with each other. They have their nuances and have to be isolated as well.

4.11.1. Tracee signals

The tracee can signal only to the visible processes and those are limited by the PID namespace. However, it can also send signals to its process group that can span multiple PID namespaces. Figure 4.1 illustrates this situation. Therefore it is necessary to set new process group for the tracee processes. Furthermore, as a hardening, a new process group is set for the PID 1 process as well, in case the tracee takes control of it.

However, it is better to also set a new session id using `setsid` syscall instead of just the process group id to avoid vulnerabilities connected to the current controlling terminal [4].

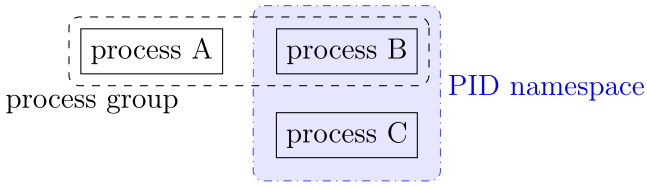

Figure 4.1: Process group can span multiple PID namespaces.

4.11.2. `SIGPIPE` in the supervisor process

Sending response to the client may generate `SIGPIPE` signal for the supervisor process if the client cancels the request approximately in the same moment. We have to ignore this signal in the supervisor process. However, this cannot be done using `SIG_IGN` because this signal disposition is not reset upon `execveat` system call and it had to be reset manually. As an alternative, it was chosen to install an empty signal handler for `SIGPIPE` so that the disposition of the signal handler is reset upon `execveat` in the tracee process automatically by the kernel.

4.11.3. Undefined Behavior Sanitizer

The code of the sandbox may be compiled with the Undefined Behavior Sanitizer (UBSan) enabled. UBSan installs signal handlers for `SIGBUS`, `SIGFPE` and `SIGSEGV` signals. This is problematic, because the tracee could send these signals to the PID 1 process. For the init process in the PID namespace, the kernel only allows sending signals for which the init process (here the PID 1 process) has installed the signal handlers [33]. Therefore the PID 1 process resets signal dispositions of these handlers if the UBSan is used to prevent the tracee from sending these signals to the PID 1 process.

4.12. Running as superuser

The sandbox is not safe to be run by the superuser. If this is needed, then you have to switch to some unprivileged user first. This is because many global system resources are still available, even after dropping the capabilities, e.g. rising privileges works. The check is done in the supervisor process at startup. To make it user namespace-proof it is checked if `/dev/null` is the null device and if the effective user id of the process equals the owner of the `/dev/null` file.

4.13. Performance optimizations

Everything that can be done is done in the supervisor process at startup, before handling requests e.g. creating cgroups, entering the network namespace, opening `/dev/null` as standard file descriptors. Sharing this work between requests ensures minimal overhead of handling the request i.e. it increases throughput (handled requests per second). Some of the optimizations are described in this section.

4.13.1. Seccomp filter of the PID 1 process

The seccomp filter of the PID 1 process is created and compiled in the supervisor process. Therefore it is done once instead of for every request.

4.13.2. Seccomp filter as file descriptor

The seccomp filter in a request is sent as a file descriptor. This avoids unnecessary copies of the seccomp filter contents in case the filter is large. Moreover, the gain is more evident if the same filter is used for subsequent requests.

4.13.3. Unsharing network, ipc, uts and time namespace

Unsharing of network, ipc, uts and time namespace is done in the supervisor process, only once, at startup. This avoids doing it for every request in the PID 1 process and has non-negligible impact on the performance (see Chapter 5).

4.14. Integration with Online Judge Platform

To integrate the new sandbox with the Online Judge Platform, a suite for each language was needed. The suite sandboxes the compiler if the language is compiled and sandboxes the runtime of the tested program. The following suites were implemented:

- C, C++, Pascal and Rust — fully compiled languages, the suite has to sandbox the compilation process and a fully compiled executable.
- Python, Bash — fully interpreted languages, the suite does not have a compilation stage, but requires sandboxing the interpreter when it runs the solution.

The Bash language is used only for testing due to its short start-up time.

Each of the compilation and run stages requires creating a root file system and a seccomp BPF filter. Root file system has to include the following bind mounts (due to dynamically linked executables):

- /lib
- /lib64
- /usr/lib
- /usr/lib64

Additionally, C, and C++ compilers require `/usr/bin` and `/usr/include`. Pascal compiler requires `/usr/bin` and `/tmp`. Rust compiler requires `/usr/bin`, `/tmp`, and on Debian `/proc`. Bash and Python require no additional bind mounts.

4.14.1. Interactive problems

Interactive problems require the tested program to communicate with the checker program i.e. the standard input of the tested program is the standard output of the checker program and the standard output of the tested program is the standard input of the checker program. Figure 4.2 illustrates this configuration. To accomplish this we need two pipes, one for each communication channel. However, the judge needs to know which process died first to provide a reasonable verdict of checking the tested program on the test.

To see why, lets consider the two examples. In the first one, the checker decides early that the tested program answered wrong, it terminates with a message "Wrong answer". Then, the pipe closes and the tested program may get terminated by `SIGPIPE` for trying to write to the closed pipe. If this happens, the tested program's abnormal death is caused by the checker exiting early. In this situation verdict "Wrong answer" is the expected verdict. In the second example, an incorrect tested program terminates early and abnormally. In this case, the pipe closes after the tested program's death and the checker sees the output of the tested program as incomplete and decides "Wrong answer". However, in this example an expected verdict would be "Runtime error" because the tested program's death caused checker to decide "Wrong answer", therefore the tested program's abnormal death takes precedence here. If we don't know who died first in such cases, we cannot reliably deduce the primary cause and therefore cannot decide what is more important, a the tested program's abnormal death or the checker's verdict.

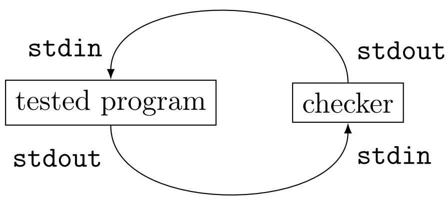

Figure 4.2: Schema of the communication between the tested program and the checker in the interactive problem.

To solve this, one could monitor the ends of the two pipes and see which end closes first. This is possible with e.g. `poll` syscall. However, it is prone to a race condition because the process monitoring the ends of the pipes may be scheduled after the checker and the tested program process and see them as if they died at the same moment. For example, the tested program process dies abnormally, then checker decides "Wrong answer" and exits and only then the `poll` syscall returns reporting all ends of the pipes as closed without the information which closed first. To avoid this race condition and decide reliably 4 pipes are needed and a process that glues both pairs of pipes together and detects which end is closed first. Figure 4.3 illustrates this configuration. As long as, the third process holds open inner four ends of the pipe pairs, the tested program and the checker will not see their `stdin` and `stdout` as broken and will not proceed (to terminate, either normally or not). This way we can reliably detect who dies first and give a correct verdict in the scenarios where one's death causes the other's death. To efficiently pass messages in the third process, the `splice` syscall is used.

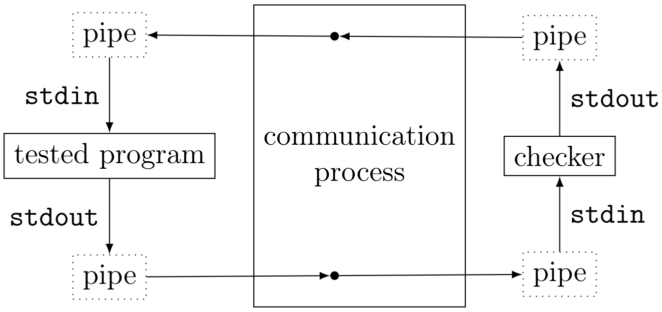

Figure 4.3: This is how communication between the tested program and the checker is implemented. Two pairs of pipes are used. This allows detection which process dies first — the tested program or the checker. Because the communication process does not close its ends of the pipes first, it can detect which process died first without causing the second to die because of a broken pipe. To efficiently pass messages in the communication process, the `splice` syscall is used.

4.14.2. Non-interactive problems

In the non-interactive problems, the semantics of the input is read-once and of the output is write-once. To achieve this without disallowing `dup`'ing, `close`'ing, `mmap`'ping, `pread`'ing etc. of the standard input and output file descriptors, we use pipes that are read-once and write-once. Input file is piped to `stdin` of the tested program, and the tested program's output is piped to the output file. Figure 4.4 illustrates this configuration. To efficiently pass messages between a pipe and a file the `splice` syscall is used.

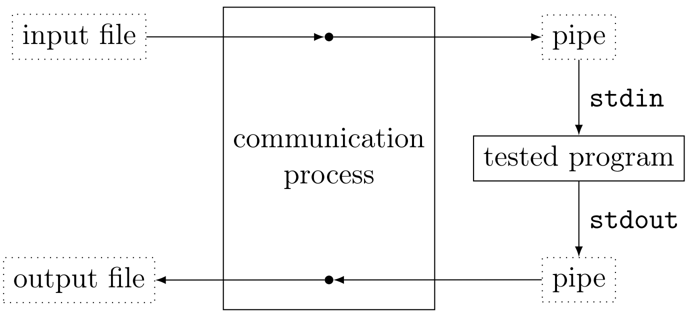

Figure 4.4: To provide the read-once semantics of the standard input and the write-once semantics of the standard output the pipes are used. The communication process passes contents of the file to the input pipe that outputs to the tested program's `stdin`. The tested program's `stdout` is piped to the output file. To efficiently pass data between files and pipes in the communication process, the `splice` syscall is used.

4.14.3. Conclusion

Apart from the above difficulties, the integration was rather easy. It required changing usage of the old sandbox to the new sandbox knobs. A lot of code was simplified along the way.

4.15. Testing and validation

To test and validate the new sandbox a comprehensive set of unit tests was developed. Tests check that namespaces are used appropriately, the interface is implemented correctly, the limits are enforced, runtime statistics are provided and are correct, no file descriptor leaks to the traced process, etc. Error reporting was also tested e.g. that unexpected supervisor death is reported, the supervisor terminates as soon as the socket connection with the client becomes broken etc. These tests ensured no regressions during the development and in the end eased the development process.

4.16. Challenges faced

There were many challenges during the development of the sandbox. First was to understand the semantics of the kernel's interfaces i.e. namespaces, cgroups and capabilities, and nuances in every one of them. One of the great achievements was to desist from using `ptrace` and all its complexity, while controlling a group of processes. This reduced the overhead and simplified things a lot. It was possible thanks to the Linux namespaces and cgroups.

Another challenge was to orchestrate everything to work together: resource limits, file descriptors, communication between processes, setting up namespaces and cgroups, dropping capabilities etc. It all has to be done in the right order and was often unobvious how to do it right.

The hardest of all was to debug very obscure errors happening during setup of the namespaces and cgroups. Often configuration failed with `EPERM` or `EINVAL` and it required figuring out what was wrong with just such vague information. For example, mounting cgroup2 file system is not allowed without unsharing cgroup namespace first. It also requires unsharing mount namespace and user namespace, but this is completely reasonable. In Sections 4.16.1 and 4.16.2 two examples are shown of the hard to debug errors.

4.16.1. `execveat` returns `EINVAL`

Executing a dynamically linked executable may fail with an error `EINVAL`. The executable needs the dynamic linker, so one reason of the error is the missing dynamic linker in the new root file system, on x86_64 it is at `/lib64/ld-linux-x86-64.so.2`. Another reason may be a missing shared library that usually resides in the directory `/usr/lib/`. It is important to bind mount all of these paths during creation of the root file system for the dynamic executables to work.

4.16.2. `execveat` returns `ENOENT`

Executing a file may fail with `ENOENT`, even though the file exists. This may happen because the file is a symbolic link and the destination does not exist, or if the symbolic link is recursive (refers to a symbolic link) and one of the intermediate files is non-existent in the new root file system. One of the solutions is to bind mount the executable without `AT_SYMLINK_NOFOLLOW`.

4.17. Conclusion

Despite challenges and complexity the sandbox was implemented and tested successfully. The usage of Linux namespaces provides isolation while cgroups and prlimits limit the resources. The sandbox is versatile enough to be used both as a sandbox for running a tested program as well as for running the compiler. The goal of optimizing the implementation for short-running programs was achieved with several optimizations.

Chapter 5

Performance Evaluation

All performance tests were made on a laptop with Intel i5-8250U processor and 16 GB of RAM. The purpose of this chapter is to verify the efficiency of the sandbox in the context of the online judge platform and briefly compare it with nsjail.

5.1. In the context of the online judge

For the testing we use tasks from the finale of XXII Polish Olimpiad in Informatics: Myjnie (myj), Tablice kierunkowe (tab), Modernizacja autostrady (mod), Wycieczki (wyc). The people who prepare the problem package for the Olympiad write different solutions programs as part of creating the problem package. The solution programs are used to verify the test cases will distinguish the different solutions of the users e.g. those running in $O(n)$ and $O(n^2)$. The model solution program is the best solution program for the problem. In Polish it is called "rozwiązanie wzorcowe". The term "solution" can be misleading but is currently used in the English literature regarding Olympiads in Informatics [12, 20, 23, 24].

First, we compare the compilation times of all solution programs outside sandbox, inside sandbox and inside sandbox with BPF filters disabled. Then we compare the running times outside the sandbox and inside the sandbox of the model solution program of each problem.

5.1.1. Compilation

Table A.1 contains the compilation times of the solution programs of the problem Myjnie. Table A.2 contains the compilation times of the solution programs of the problem Tablice kierunkowe. Table A.3 contains the compilation times of the solution programs of the problem Modernizacja autostrady. Table A.4 contains the compilation times of the solution programs of the problem Wycieczki.

From the measurements we see that most of the overhead comes from BPF filters that cause up to 10% slowdown. The compilation in the sandbox can be slower by up to 24%, but can be also be faster by up to 12%. We could not identify where the speedup comes from, it requires further investigation. Most of the time, the sandbox's slowdown is below 10% compared to compilation time without the sandbox.

Commands used in testing:

- C — `/usr/bin/gcc -std=c11 -O2 -static -o exe source.c`. Compiler version: 13.2.1.
- C++ — `/usr/bin/g++ -std=c++17 -O2 -static -o exe source.cc`. Compiler version: 13.2.1.

- Pascal — `/usr/bin/fpc -O2 -XS -Xt -oexe source.pas`. Compiler version 3.3.2.

5.1.2. Model solutions' run times

Table A.5 contains the run times of the model solution program of problem Myjnie. Table A.6 contains the run times of the model solution program of problem Tablice kierunkowe. Table A.7 contains the run times of the model solution program of problem Modernizacja autostrady. Table A.8 contains the run times of the model solution program of problem Wycieczki.

From the measurements we see that on large tests (those with high runtime) the slowdown in the sandbox can be as high as 20%. On smaller tests there is often a speed up but it can be caused by a different method of measuring runtime i.e. `perf stat` vs. sandbox i.e. real time timer and cgroup `cpu.stat` file. We could not identify the cause of the speedup on the large tests up to 15%.

5.2. Short-running programs and comparison with nsjail

An example of a short-running program is `/bin/true` — we will use it in the benchmark. The metric we will use is the round-trip time of the request to sandbox a program. Table 5.1 contains the measurements of the time to handle request to run the `/bin/true` program, without the sandbox, with sandbox and using nsjail. From the data we see that the nsjail is more than 4 times slower, while at the same time it does not spawn a separate PID 1 process and does not provide the runtime statistics. Our sandbox, while 2.39 times slower, still allows for hundreds requests per second — and that was the goal of the thesis.

Sandbox	Mean time	Std. dev.	Std. err. on the mean	Slowdown
no sandbox	0.893ms	0.409ms (45.80%)	0.013ms (1.45%)	1x
sandbox	2.348ms	0.768ms (32.71%)	0.024ms (1.03%)	2.39x
nsjail	10.393ms	1.327ms (12.77%)	0.042ms (0.40%)	10.57x

Table 5.1: Statistics for each row were collected from 1000 runs. Each row contains real time it took to handle request to sandbox the `/bin/true` program. While the slowdown of the sandbox is huge (more than twofold), it still allows for hundreds of runs per second and that was the goal of this thesis, whereas nsjail is more than 4 times slower than our sandbox.

nsjail does not provide any means to handle more than one program in one execution. This means that one execution of the nsjail program equals one execution of the sandboxed program. This way, the nsjail program cannot share resources e.g. namespaces between the runs and therefore should be slower than our solution. Our solution executes the server program once and it can handle more than one request for secure execution of a program therefore it can and shares the resources. This way, our solution has lower overhead per one execution of the sandboxed program. Therefore the request handle time is drastically lower than for our sandbox. This is presented in Table 5.1. The nsjail can handle around 4.4 times less requests than our solution in a given time. Moreover, our sandbox allows easy collection of the runtime statistics that nsjail is incapable of doing. Command used for benchmarking the nsjail: `/usr/bin/nsjail -q -Mo --chroot / --use_cgroupv2 -- /bin/true`.

5.3. Impact of some optimizations

From the Table 5.2 it is clear that unsharing namespaces once instead of for every request has positive impact on the performance. The most meaningful was the network namespace that if unshared for every request resulted in 26% performance degradation.

Benchmark	Mean request time	Std. dev.	Std. err. on the mean	Slowdown
Baseline	2.348ms	0.768ms (32.71%)	0.024ms (1.03%)	0.00%
New network namespace for each request	2.970ms	0.856ms (28.83%)	0.027ms (0.91%)	26.49%
New IPC namespace for each request	2.522ms	0.782ms (31.02%)	0.025ms (0.98%)	7.41%
New UTS namespace for each request	2.478ms	0.771ms (31.14%)	0.024ms (0.98%)	5.54%

Table 5.2: Statistics for each row were collected from 1000 runs. Each row contains real time it took to handle request to sandbox the `/bin/true` program.

5.4. Conclusion

Although the slowdown of the compilation of around 24% and run time of the solution program of around 20% is noticeable, it is acceptable. More importantly, the overhead of running the tested program is more often negligible than not. Experiments showed that most of the compilation overhead is caused by BPF filters, but they are required for the security of the sandbox. The sandbox allows handling hundreds of requests per second. The concurrent solution — nsjail is 4.4 times slower than ours. All in all, the goal of the thesis was achieved.

Chapter 6

Conclusion

The sandbox was successfully designed, implemented and integrated with the online judge platform. A new language suites — for Python and Rust were implemented using it.

The new sandbox is optimized for running short-running programs and the overhead of running one in below 3 ms. That is 4 times lower than the closest solution — nsjail. The sandbox is versatile enough to sandbox simple user programs as well as the full compilation. The overhead of running the program in the sandbox is below 24% and often it is even lower than 0%.

The sandbox allows limiting resources and collects the run-time statistics of the executed program. This is a very unique feature of our solution.

Furthermore, our sandbox does not require any privileges — it can be used by any unprivileged user.

All in all, the goal of the thesis was achieved although with higher overheads than anticipated. However, the overhead is within the acceptable margin.

6.1. Future work

There are several aspects that can be worked upon. Each of a different degree of difficulty.

6.1.1. CPU affinity

Now, there is no support for setting CPU affinity mask for the request. Such support should be straightforward to add. It could reduce time variability of subsequent runs of the tested program [23].

6.1.2. Adding support for networking

Although disabling the networking altogether is secure, adding a loopback device can be beneficial for some applications. Adding other devices requires superuser privileges and is out-of-scope for the current solution, but can be done with a `setuid` helper binary like in Firejail [25].

6.1.3. Rust frontend

Implementing the client side (a frontend) in Rust language should ease the adoption of the sandbox. Rust is an efficient and secure programming language. In Rust a simple cargo package with the frontend could be implemented and published to the world. Such a package would be easy to use and allow easy and quick experimenting with the sandbox.

6.1.4. Further experimentation

Investigating from where the speedups come from and why there are such big slowdowns despite BPF being disabled are what can be improved upon. Such knowledge would provide valuable insight into the situation.

6.2. Acknowledgements

We would like to thank Janina Mincer-Daszkiewicz for relentless and valuable insight during the writing of the thesis.

Bibliography

[1] David Beserra et al. “Performance Analysis of LXC for HPC Environments.” In: *CISIS*. IEEE Computer Society, 2015, pp. 358–363. ISBN: 978-1-4799-8870-9. URL: http://dblp.uni-trier.de/db/conf/cisis/cisis2015.html#BeserraMEBSF15.

[2] Sander van der Burg and Eelco Dolstra. “Automating System Tests Using Declarative Virtual Machines”. In: *2010 IEEE 21st International Symposium on Software Reliability Engineering*. 2010, pp. 181–190. DOI: 10.1109/ISSRE.2010.34.

[3] Henrique Zanela Cochak et al. “RunC and Kata runtime using Docker: a network perspective comparison”. In: *2021 IEEE Latin-American Conference on Communications (LATINCOM)*. IEEE. 2021, pp. 1–6.

[4] *CVE-2017-5226 – Bubblewrap escape*. URL: https://security-tracker.debian.org/tracker/CVE-2017-5226 (visited on 2023-10-22).

[5] *CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition*. URL: https://cwe.mitre.org/data/definitions/367.html (visited on 2023-10-19).

[6] Flatpak. *Flatpak - the future of application distribution*. URL: https://flatpak.org/ (visited on 2023-10-17).

[7] T Garfinkel. “Janus: A practical tool for application sandboxing”. In: *http://www. cs. berkeley. edu/daw/janus* (2004).

[8] Tal Garfinkel, Ben Pfaff, Mendel Rosenblum, et al. “Ostia: A Delegating Architecture for Secure System Call Interposition.” In: *NDSS*. 2004.

[9] Google. *A light-weight process isolation tool, making use of Linux namespaces and seccomp-bpf syscall filters*. URL: https://github.com/google/nsjail (visited on 2023-10-17).

[10] Suman Jana, Donald E Porter, and Vitaly Shmatikov. “TxBox: Building secure, efficient sandboxes with system transactions”. In: *2011 IEEE Symposium on Security and Privacy*. IEEE. 2011, pp. 329–344.

[11] Taesoo Kim and Nickolai Zeldovich. “Practical and effective sandboxing for non-root users”. In: *2013 USENIX Annual Technical Conference (USENIX ATC 13)*. 2013, pp. 139–144.

[12] Rob Kolstad. “Infrastructure for contest task development”. In: *Olympiads in Informatics* 3 (2009), pp. 38–59.

[13] Yanlin Li et al. “{MiniBox}: A {Two-Way} Sandbox for x86 Native Code”. In: *2014 USENIX annual technical conference (USENIX ATC 14)*. 2014, pp. 409–420.

[14] *Low-level unprivileged sandboxing tool used by Flatpak and similar projects*. URL: https://github.com/containers/bubblewrap (visited on 2023-10-19).

[15] Redis Ltd. *Diagnosing latency issues: Latency generated by fork.* 2011-09-08. URL: https://redis.io/docs/reference/optimization/latency/#latency-generated-by-fork (visited on 2022-09-08).

[16] Krzysztof Małysa. *Sim project.* URL: https://github.com/varqox/sim (visited on 2023-03-15).

[17] Krzysztof Małysa. *Sip – a tool for preparing problem packages for the Sim platform.* URL: https://github.com/varqox/sip (visited on 2023-03-15).

[18] Martin Mareš. "Fairness of Time Constraints." In: *Olympiads in Informatics* 5 (2011), pp. 92–102.

[19] Martin Mareš. "Perspectives on grading systems". In: *Olympiads in Informatics* (2007), pp. 124–130.

[20] Martin Mareš and Bernard Blackham. "A New Contest Sandbox." In: *Olympiads in Informatics* 6 (2012), pp. 100–109. URL: https://ioi.te.lv/oi/pdf/INFOL094.pdf.

[21] Jonathan M McCune et al. "TrustVisor: Efficient TCB reduction and attestation". In: *2010 IEEE Symposium on Security and Privacy*. IEEE. 2010, pp. 143–158.

[22] Dirk Merkel. "Docker: Lightweight Linux Containers for Consistent Development and Deployment". In: *Linux J.* 2014.239 (2014-03). ISSN: 1075-3583. URL: http://dl.acm.org/citation.cfm?id=2600239.2600241.

[23] Bruce Merry. "Performance analysis of sandboxes for reactive tasks". In: *Olympiads in Informatics* 4 (2010), pp. 87–94.

[24] Bruce Merry. "Using a Linux security module for contest security". In: *Olympiads in Informatics* 3 (2009), pp. 67–73.

[25] netblue30/firejail. *Linux namespaces and seccomp-bpf sandbox.* URL: https://github.com/netblue30/firejail (visited on 2023-10-17).

[26] *Official website of Kernel Virtual Machine.* URL: https://www.linux-kvm.org/ (visited on 2022-11-23).

[27] *Official website of QEMU — A generic and open source machine emulator and virtualizer.* URL: https://www.qemu.org/ (visited on 2022-11-23).

[28] Oracle. *Official website of VirtualBox.* URL: https://www.virtualbox.org/ (visited on 2022-11-23).

[29] Vassilis Prevelakis and Diomidis Spinellis. "Sandboxing Applications." In: *Usenix annual technical conference, freenix track.* Citeseer. 2001, pp. 119–126.

[30] man-pages project. *execveat - execute program relative to a directory file descriptor.* URL: https://man7.org/linux/man-pages/man2/execveat.2.html (visited on 2023-10-20).

[31] man-pages project. *getrlimit, setrlimit, prlimit - get/set resource limits.* URL: https://man7.org/linux/man-pages/man2/prlimit.2.html (visited on 2023-10-18).

[32] man-pages project. *mount_namespaces - overview of Linux mount namespaces.* URL: https://man7.org/linux/man-pages/man7/mount_namespaces.7.html (visited on 2023-10-20).

[33] man-pages project. *pid_namespaces - overview of Linux PID namespaces.* URL: https://man7.org/linux/man-pages/man7/pid_namespaces.7.html (visited on 2022-11-13).

[34] man-pages project. *setuid - set user identity.* URL: https://man7.org/linux/man-pages/man2/setuid.2.html (visited on 2023-10-18).

[35] man-pages project. *unix - sockets for local interprocess communication*. URL: https://man7.org/linux/man-pages/man7/unix.7.html (visited on 2023-10-20).

[36] Niels Provos. "Improving Host Security with System Call Policies." In: *USENIX Security Symposium*. 2003, pp. 257–272.

[37] Inge Alexander Raknes, Bjørn Fjukstad, and Lars Ailo Bongo. "nsroot: Minimalist process isolation tool implemented with linux namespaces". In: *arXiv preprint arXiv:1609.03750* (2016).

[38] rootlesscontaine.rs. *Rootless Containers*. URL: https://rootlesscontaine.rs (visited on 2022-11-28).

[39] Giuseppe Scrivano. *Rootless containers with Podman and fuse-overlayfs*. 2019-06-04. URL: https://indico.cern.ch/event/757415/contributions/3421994/attachments/1855302/3047064/Podman_Rootless_Containers.pdf (visited on 2022-11-28).

[40] František Špaček, Radomír Sohlich, and Tomáš Dulík. "Docker as Platform for Assignments Evaluation". In: *Procedia Engineering* 100 (2015). 25th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2014, pp. 1665–1671. ISSN: 1877-7058. DOI: https://doi.org/10.1016/j.proeng.2015.01.541. URL: https://www.sciencedirect.com/science/article/pii/S1877705815005688.

[41] *strace - trace system calls and signals*. URL: https://man7.org/linux/man-pages/man1/strace.1.html (visited on 2023-10-20).

[42] systemd. *systemd-nspawn — Spawn a command or OS in a light-weight container*. URL: https://www.freedesktop.org/software/systemd/man/systemd-nspawn.html (visited on 2022-11-28).

[43] Tocho Tochev and Tsvetan Bogdanov. "Validating the Security and Stability of the Grader for a Programming Contest System." In: *Olympiads in Informatics* 4 (2010), pp. 113–119.

[44] VMWare. *Official website of VMWare Workstation*. URL: https://www.vmware.com/products/workstation/ (visited on 2022-11-23).

[45] Bennet Yee et al. "Native client: A sandbox for portable, untrusted x86 native code". In: *Communications of the ACM* 53.1 (2010), pp. 91–99.

Appendices

Appendix A

Tables

A.1. Compilation times

A.1.1. Myjnie (myj)

Solution program	Sandbox	Time	Mean	Std. dev.	Std. err. on the mean	Slowdown
myj.cpp	no	real	362.60ms	8.85ms (2.44%)	2.80ms (0.77%)	0.00%
		CPU	356.96ms	5.01ms (1.40%)	1.59ms (0.44%)	0.00%
	yes	real	386.11ms	7.27ms (1.88%)	2.30ms (0.60%)	6.48%
		CPU	379.41ms	6.12ms (1.61%)	1.93ms (0.51%)	6.29%
	no BPF	real	359.60ms	6.73ms (1.87%)	2.13ms (0.59%)	-0.83%
		CPU	350.94ms	5.31ms (1.51%)	1.68ms (0.48%)	-1.69%
myj1.pas	no	real	72.18ms	6.26ms (8.68%)	1.98ms (2.74%)	0.00%
		CPU	69.66ms	4.93ms (7.07%)	1.56ms (2.24%)	0.00%
	yes	real	70.10ms	4.26ms (6.07%)	1.35ms (1.92%)	-2.87%
		CPU	67.60ms	3.52ms (5.21%)	1.11ms (1.65%)	-2.96%
	no BPF	real	69.68ms	5.00ms (7.17%)	1.58ms (2.27%)	-3.46%
		CPU	67.99ms	5.09ms (7.48%)	1.61ms (2.37%)	-2.39%
myj2.cpp	no	real	369.42ms	7.44ms (2.01%)	2.35ms (0.64%)	0.00%
		CPU	362.72ms	4.64ms (1.28%)	1.47ms (0.40%)	0.00%
	yes	real	388.42ms	8.44ms (2.17%)	2.67ms (0.69%)	5.14%
		CPU	377.39ms	6.00ms (1.59%)	1.90ms (0.50%)	4.04%
	no BPF	real	352.86ms	5.79ms (1.64%)	1.83ms (0.52%)	-4.48%
		CPU	348.17ms	5.48ms (1.57%)	1.73ms (0.50%)	-4.01%
myj3.cpp	no	real	394.17ms	6.51ms (1.65%)	2.06ms (0.52%)	0.00%
		CPU	384.30ms	5.93ms (1.54%)	1.88ms (0.49%)	0.00%
	yes	real	404.30ms	6.35ms (1.57%)	2.01ms (0.50%)	2.57%
		CPU	396.11ms	5.26ms (1.33%)	1.66ms (0.42%)	3.07%
	no BPF	real	378.24ms	8.30ms (2.20%)	2.63ms (0.69%)	-4.04%
		CPU	373.52ms	5.86ms (1.57%)	1.85ms (0.50%)	-2.81%
myj4.cpp	no	real	2105.00ms	32.27ms (1.53%)	10.20ms (0.48%)	0.00%
		CPU	2083.61ms	29.28ms (1.41%)	9.26ms (0.44%)	0.00%
	yes	real	2244.82ms	184.78ms (8.23%)	58.43ms (2.60%)	6.64%
		CPU	2222.96ms	183.39ms (8.25%)	57.99ms (2.61%)	6.69%
	no BPF	real	2093.29ms	39.94ms (1.91%)	12.63ms (0.60%)	-0.56%
		CPU	2069.41ms	32.68ms (1.58%)	10.33ms (0.50%)	-0.68%

myjb1.cpp	no	real	366.94ms	7.57ms (2.06%)	2.40ms (0.65%)	0.00%
		CPU	359.89ms	4.60ms (1.28%)	1.46ms (0.40%)	0.00%
	yes	real	384.46ms	6.89ms (1.79%)	2.18ms (0.57%)	4.77%
		CPU	376.95ms	5.20ms (1.38%)	1.65ms (0.44%)	4.74%
	no BPF	real	353.91ms	8.68ms (2.45%)	2.74ms (0.78%)	-3.55%
		CPU	346.75ms	4.14ms (1.19%)	1.31ms (0.38%)	-3.65%
myjb2.cpp	no	real	334.10ms	3.68ms (1.10%)	1.16ms (0.35%)	0.00%
		CPU	329.16ms	4.85ms (1.47%)	1.53ms (0.47%)	0.00%
	yes	real	336.05ms	6.57ms (1.96%)	2.08ms (0.62%)	0.58%
		CPU	328.03ms	4.88ms (1.49%)	1.54ms (0.47%)	-0.34%
	no BPF	real	321.29ms	6.00ms (1.87%)	1.90ms (0.59%)	-3.84%
		CPU	314.97ms	5.34ms (1.70%)	1.69ms (0.54%)	-4.31%
myjb3.cpp	no	real	369.83ms	6.01ms (1.63%)	1.90ms (0.51%)	0.00%
		CPU	365.12ms	7.22ms (1.98%)	2.28ms (0.63%)	0.00%
	yes	real	380.30ms	6.99ms (1.84%)	2.21ms (0.58%)	2.83%
		CPU	373.70ms	2.52ms (0.67%)	0.80ms (0.21%)	2.35%
	no BPF	real	351.72ms	3.58ms (1.02%)	1.13ms (0.32%)	-4.90%
		CPU	346.07ms	3.65ms (1.06%)	1.16ms (0.33%)	-5.22%
myjb4.cpp	no	real	355.35ms	8.39ms (2.36%)	2.65ms (0.75%)	0.00%
		CPU	348.56ms	6.47ms (1.86%)	2.05ms (0.59%)	0.00%
	yes	real	364.72ms	6.54ms (1.79%)	2.07ms (0.57%)	2.64%
		CPU	358.57ms	6.47ms (1.80%)	2.04ms (0.57%)	2.87%
	no BPF	real	343.04ms	7.69ms (2.24%)	2.43ms (0.71%)	-3.46%
		CPU	333.60ms	3.65ms (1.09%)	1.16ms (0.35%)	-4.29%
myjb5.cpp	no	real	358.90ms	8.32ms (2.32%)	2.63ms (0.73%)	0.00%
		CPU	349.96ms	8.32ms (2.38%)	2.63ms (0.75%)	0.00%
	yes	real	370.24ms	8.78ms (2.37%)	2.78ms (0.75%)	3.16%
		CPU	362.40ms	4.99ms (1.38%)	1.58ms (0.44%)	3.55%
	no BPF	real	342.85ms	7.98ms (2.33%)	2.52ms (0.74%)	-4.47%
		CPU	336.04ms	5.80ms (1.72%)	1.83ms (0.55%)	-3.98%
myjs1.cpp	no	real	336.79ms	31.10ms (9.23%)	9.83ms (2.92%)	0.00%
		CPU	332.67ms	29.36ms (8.83%)	9.29ms (2.79%)	0.00%
	yes	real	338.45ms	35.76ms (10.57%)	11.31ms (3.34%)	0.49%
		CPU	330.57ms	28.16ms (8.52%)	8.91ms (2.69%)	-0.63%
	no BPF	real	356.88ms	6.26ms (1.75%)	1.98ms (0.55%)	5.97%
		CPU	348.74ms	6.58ms (1.89%)	2.08ms (0.60%)	4.83%
myjs2.pas	no	real	69.89ms	4.75ms (6.80%)	1.50ms (2.15%)	0.00%
		CPU	68.40ms	4.49ms (6.57%)	1.42ms (2.08%)	0.00%
	yes	real	69.28ms	5.55ms (8.02%)	1.76ms (2.53%)	-0.86%
		CPU	66.16ms	5.26ms (7.95%)	1.66ms (2.51%)	-3.28%
	no BPF	real	67.74ms	5.00ms (7.39%)	1.58ms (2.34%)	-3.07%
		CPU	65.72ms	4.25ms (6.46%)	1.34ms (2.04%)	-3.92%
myjs3.cpp	no	real	369.35ms	7.23ms (1.96%)	2.29ms (0.62%)	0.00%
		CPU	360.47ms	6.47ms (1.79%)	2.04ms (0.57%)	0.00%
	yes	real	380.47ms	7.97ms (2.09%)	2.52ms (0.66%)	3.01%
		CPU	371.12ms	6.12ms (1.65%)	1.93ms (0.52%)	2.95%
	no BPF	real	354.97ms	7.63ms (2.15%)	2.41ms (0.68%)	-3.89%
		CPU	345.67ms	6.99ms (2.02%)	2.21ms (0.64%)	-4.11%
myjs4.pas	no	real	72.02ms	4.68ms (6.50%)	1.48ms (2.05%)	0.00%
		CPU	70.75ms	4.86ms (6.87%)	1.54ms (2.17%)	0.00%
	yes	real	69.31ms	4.13ms (5.95%)	1.30ms (1.88%)	-3.77%
		CPU	67.67ms	3.94ms (5.82%)	1.25ms (1.84%)	-4.35%
	no BPF	real	61.85ms	7.01ms (11.34%)	2.22ms (3.59%)	-14.13%
		CPU	60.24ms	6.41ms (10.65%)	2.03ms (3.37%)	-14.86%

myjs5.cpp	no	real	310.64ms	23.84ms (7.67%)	7.54ms (2.43%)	0.00%
		CPU	306.25ms	17.21ms (5.62%)	5.44ms (1.78%)	0.00%
	yes	real	384.98ms	7.94ms (2.06%)	2.51ms (0.65%)	23.93%
		CPU	380.27ms	7.79ms (2.05%)	2.46ms (0.65%)	24.17%
	no BPF	real	358.36ms	7.67ms (2.14%)	2.43ms (0.68%)	15.36%
		CPU	351.52ms	4.90ms (1.40%)	1.55ms (0.44%)	14.78%
myjs6.pas	no	real	71.69ms	4.21ms (5.88%)	1.33ms (1.86%)	0.00%
		CPU	70.00ms	4.01ms (5.73%)	1.27ms (1.81%)	0.00%
	yes	real	68.40ms	4.09ms (5.98%)	1.29ms (1.89%)	-4.58%
		CPU	66.64ms	3.65ms (5.48%)	1.15ms (1.73%)	-4.81%
	no BPF	real	69.24ms	4.16ms (6.00%)	1.31ms (1.90%)	-3.41%
		CPU	66.45ms	3.33ms (5.02%)	1.05ms (1.59%)	-5.08%
myjs7.cpp	no	real	2123.70ms	40.65ms (1.91%)	12.85ms (0.61%)	0.00%
		CPU	2109.31ms	33.11ms (1.57%)	10.47ms (0.50%)	0.00%
	yes	real	2451.10ms	37.92ms (1.55%)	11.99ms (0.49%)	15.42%
		CPU	2425.73ms	28.52ms (1.18%)	9.02ms (0.37%)	15.00%
	no BPF	real	2131.85ms	35.95ms (1.69%)	11.37ms (0.53%)	0.38%
		CPU	2111.61ms	36.11ms (1.71%)	11.42ms (0.54%)	0.11%
myjs8.cpp	no	real	2013.38ms	150.21ms (7.46%)	47.50ms (2.36%)	0.00%
		CPU	1997.83ms	149.04ms (7.46%)	47.13ms (2.36%)	0.00%
	yes	real	2411.73ms	33.09ms (1.37%)	10.46ms (0.43%)	19.79%
		CPU	2382.87ms	31.90ms (1.34%)	10.09ms (0.42%)	19.27%
	no BPF	real	2070.14ms	38.98ms (1.88%)	12.33ms (0.60%)	2.82%
		CPU	2053.40ms	38.97ms (1.90%)	12.32ms (0.60%)	2.78%

Table A.1: Compilation times of all solution programs of the problem Myjnie (myj) from the finale of XXII Polish Olimpiad in Informatics. For each configuration (Solution program and Sandbox columns) the data was collected from 10 runs. Real and CPU times were collected from the same runs. Slowdown is measured from the times of configuration without the sandbox.

A.1.2. Tablice kierunkowe (tab)

Solution program	Sandbox	Time	Mean	Std. dev.	Std. err. on the mean	Slowdown
tab.cpp	no	real	1502.95ms	25.17ms (1.67%)	7.96ms (0.53%)	0.00%
		CPU	1488.30ms	20.63ms (1.39%)	6.52ms (0.44%)	0.00%
	yes	real	1634.57ms	18.91ms (1.16%)	5.98ms (0.37%)	8.76%
		CPU	1609.73ms	19.95ms (1.24%)	6.31ms (0.39%)	8.16%
	no BPF	real	1402.92ms	105.52ms (7.52%)	33.37ms (2.38%)	-6.66%
		CPU	1389.83ms	102.23ms (7.36%)	32.33ms (2.33%)	-6.62%
tab2.c	no	real	222.52ms	79.15ms (35.57%)	25.03ms (11.25%)	0.00%
		CPU	192.33ms	14.33ms (7.45%)	4.53ms (2.36%)	0.00%
	yes	real	195.05ms	4.84ms (2.48%)	1.53ms (0.79%)	-12.34%
		CPU	190.64ms	3.50ms (1.84%)	1.11ms (0.58%)	-0.88%
	no BPF	real	196.87ms	4.45ms (2.26%)	1.41ms (0.72%)	-11.53%
		CPU	190.74ms	2.57ms (1.35%)	0.81ms (0.43%)	-0.83%

tab3.pas	no	real	75.84ms	5.48ms (7.23%)	1.73ms (2.28%)	0.00%
		CPU	72.71ms	4.38ms (6.02%)	1.38ms (1.90%)	0.00%
	yes	real	71.19ms	3.61ms (5.07%)	1.14ms (1.60%)	-6.14%
		CPU	69.66ms	3.18ms (4.56%)	1.00ms (1.44%)	-4.20%
	no BPF	real	70.14ms	5.72ms (8.16%)	1.81ms (2.58%)	-7.53%
		CPU	67.41ms	4.27ms (6.34%)	1.35ms (2.00%)	-7.30%
tab4.cpp	no	real	2261.35ms	39.40ms (1.74%)	12.46ms (0.55%)	0.00%
		CPU	2232.11ms	33.15ms (1.49%)	10.48ms (0.47%)	0.00%
	yes	real	2525.38ms	62.24ms (2.46%)	19.68ms (0.78%)	11.68%
		CPU	2503.59ms	61.71ms (2.46%)	19.51ms (0.78%)	12.16%
	no BPF	real	2182.13ms	43.47ms (1.99%)	13.75ms (0.63%)	-3.50%
		CPU	2154.98ms	35.40ms (1.64%)	11.19ms (0.52%)	-3.46%
tabb1.cpp	no	real	1631.92ms	121.04ms (7.42%)	38.28ms (2.35%)	0.00%
		CPU	1615.02ms	120.35ms (7.45%)	38.06ms (2.36%)	0.00%
	yes	real	1831.19ms	33.72ms (1.84%)	10.66ms (0.58%)	12.21%
		CPU	1813.79ms	31.32ms (1.73%)	9.90ms (0.55%)	12.31%
	no BPF	real	1679.20ms	23.03ms (1.37%)	7.28ms (0.43%)	2.90%
		CPU	1660.51ms	17.82ms (1.07%)	5.64ms (0.34%)	2.82%
tabb2.cpp	no	real	1671.93ms	25.31ms (1.51%)	8.00ms (0.48%)	0.00%
		CPU	1662.92ms	24.85ms (1.49%)	7.86ms (0.47%)	0.00%
	yes	real	1674.66ms	161.33ms (9.63%)	51.02ms (3.05%)	0.16%
		CPU	1655.46ms	150.66ms (9.10%)	47.64ms (2.88%)	-0.45%
	no BPF	real	1687.88ms	45.56ms (2.70%)	14.41ms (0.85%)	0.95%
		CPU	1664.14ms	39.50ms (2.37%)	12.49ms (0.75%)	0.07%
tabb3.cpp	no	real	2154.16ms	28.51ms (1.32%)	9.02ms (0.42%)	0.00%
		CPU	2129.23ms	26.30ms (1.24%)	8.32ms (0.39%)	0.00%
	yes	real	2471.52ms	38.72ms (1.57%)	12.24ms (0.50%)	14.73%
		CPU	2445.05ms	39.86ms (1.63%)	12.60ms (0.52%)	14.83%
	no BPF	real	2058.72ms	166.25ms (8.08%)	52.57ms (2.55%)	-4.43%
		CPU	2036.36ms	158.63ms (7.79%)	50.16ms (2.46%)	-4.36%
tabb4.cpp	no	real	1613.79ms	134.43ms (8.33%)	42.51ms (2.63%)	0.00%
		CPU	1600.23ms	130.74ms (8.17%)	41.34ms (2.58%)	0.00%
	yes	real	1883.39ms	35.67ms (1.89%)	11.28ms (0.60%)	16.71%
		CPU	1857.23ms	29.17ms (1.57%)	9.22ms (0.50%)	16.06%
	no BPF	real	1702.76ms	28.62ms (1.68%)	9.05ms (0.53%)	5.51%
		CPU	1686.87ms	27.15ms (1.61%)	8.59ms (0.51%)	5.41%
tabb5.cpp	no	real	1565.51ms	22.63ms (1.45%)	7.16ms (0.46%)	0.00%
		CPU	1557.13ms	22.90ms (1.47%)	7.24ms (0.47%)	0.00%
	yes	real	1604.59ms	143.02ms (8.91%)	45.23ms (2.82%)	2.50%
		CPU	1589.50ms	137.02ms (8.62%)	43.33ms (2.73%)	2.08%
	no BPF	real	1577.05ms	28.41ms (1.80%)	8.98ms (0.57%)	0.74%
		CPU	1558.93ms	24.23ms (1.55%)	7.66ms (0.49%)	0.12%
tabs1.cpp	no	real	2323.59ms	53.25ms (2.29%)	16.84ms (0.72%)	0.00%
		CPU	2301.03ms	50.99ms (2.22%)	16.13ms (0.70%)	0.00%
	yes	real	2570.53ms	153.29ms (5.96%)	48.47ms (1.89%)	10.63%
		CPU	2541.60ms	151.81ms (5.97%)	48.01ms (1.89%)	10.45%
	no BPF	real	2313.52ms	37.31ms (1.61%)	11.80ms (0.51%)	-0.43%
		CPU	2289.49ms	37.80ms (1.65%)	11.95ms (0.52%)	-0.50%
tabs2.cpp	no	real	1442.48ms	27.35ms (1.90%)	8.65ms (0.60%)	0.00%
		CPU	1423.01ms	16.75ms (1.18%)	5.30ms (0.37%)	0.00%
	yes	real	1568.84ms	24.78ms (1.58%)	7.84ms (0.50%)	8.76%
		CPU	1547.59ms	17.34ms (1.12%)	5.48ms (0.35%)	8.75%
	no BPF	real	1427.96ms	24.65ms (1.73%)	7.80ms (0.55%)	-1.01%
		CPU	1413.63ms	20.35ms (1.44%)	6.44ms (0.46%)	-0.66%

tabs3.cpp	no	real	1561.91ms	31.81ms (2.04%)	10.06ms (0.64%)	0.00%
		CPU	1549.61ms	31.37ms (2.02%)	9.92ms (0.64%)	0.00%
	yes	real	1695.55ms	33.16ms (1.96%)	10.49ms (0.62%)	8.56%
		CPU	1667.92ms	23.82ms (1.43%)	7.53ms (0.45%)	7.63%
	no BPF	real	1477.71ms	118.16ms (8.00%)	37.37ms (2.53%)	-5.39%
		CPU	1461.29ms	111.89ms (7.66%)	35.38ms (2.42%)	-5.70%
tabs4.cpp	no	real	2592.15ms	44.09ms (1.70%)	13.94ms (0.54%)	0.00%
		CPU	2573.68ms	38.34ms (1.49%)	12.12ms (0.47%)	0.00%
	yes	real	2822.21ms	175.59ms (6.22%)	55.53ms (1.97%)	8.88%
		CPU	2797.91ms	175.07ms (6.26%)	55.36ms (1.98%)	8.71%
	no BPF	real	2580.34ms	46.37ms (1.80%)	14.66ms (0.57%)	-0.46%
		CPU	2556.68ms	41.86ms (1.64%)	13.24ms (0.52%)	-0.66%
tabs5.cpp	no	real	2631.50ms	47.73ms (1.81%)	15.09ms (0.57%)	0.00%
		CPU	2612.73ms	38.40ms (1.47%)	12.14ms (0.46%)	0.00%
	yes	real	2862.74ms	167.67ms (5.86%)	53.02ms (1.85%)	8.79%
		CPU	2829.77ms	167.37ms (5.91%)	52.93ms (1.87%)	8.31%
	no BPF	real	2628.00ms	43.35ms (1.65%)	13.71ms (0.52%)	-0.13%
		CPU	2593.05ms	41.02ms (1.58%)	12.97ms (0.50%)	-0.75%
tabs6.cpp	no	real	1914.91ms	40.86ms (2.13%)	12.92ms (0.67%)	0.00%
		CPU	1895.37ms	34.85ms (1.84%)	11.02ms (0.58%)	0.00%
	yes	real	2106.36ms	34.78ms (1.65%)	11.00ms (0.52%)	10.00%
		CPU	2077.84ms	28.96ms (1.39%)	9.16ms (0.44%)	9.63%
	no BPF	real	1910.88ms	32.87ms (1.72%)	10.39ms (0.54%)	-0.21%
		CPU	1882.19ms	30.43ms (1.62%)	9.62ms (0.51%)	-0.70%
tabs7.cpp	no	real	2957.68ms	60.44ms (2.04%)	19.11ms (0.65%)	0.00%
		CPU	2931.50ms	55.50ms (1.89%)	17.55ms (0.60%)	0.00%
	yes	real	3282.68ms	65.61ms (2.00%)	20.75ms (0.63%)	10.99%
		CPU	3248.76ms	60.39ms (1.86%)	19.10ms (0.59%)	10.82%
	no BPF	real	2921.55ms	57.01ms (1.95%)	18.03ms (0.62%)	-1.22%
		CPU	2898.11ms	50.67ms (1.75%)	16.02ms (0.55%)	-1.14%
tabs8.cpp	no	real	1796.60ms	33.59ms (1.87%)	10.62ms (0.59%)	0.00%
		CPU	1781.45ms	25.69ms (1.44%)	8.12ms (0.46%)	0.00%
	yes	real	1946.05ms	33.29ms (1.71%)	10.53ms (0.54%)	8.32%
		CPU	1927.81ms	31.78ms (1.65%)	10.05ms (0.52%)	8.22%
	no BPF	real	1786.73ms	29.69ms (1.66%)	9.39ms (0.53%)	-0.55%
		CPU	1770.08ms	27.25ms (1.54%)	8.62ms (0.49%)	-0.64%
tabs9.cpp	no	real	2183.74ms	40.84ms (1.87%)	12.91ms (0.59%)	0.00%
		CPU	2165.15ms	40.55ms (1.87%)	12.82ms (0.59%)	0.00%
	yes	real	2404.02ms	174.72ms (7.27%)	55.25ms (2.30%)	10.09%
		CPU	2381.17ms	167.37ms (7.03%)	52.93ms (2.22%)	9.98%
	no BPF	real	2164.08ms	43.88ms (2.03%)	13.87ms (0.64%)	-0.90%
		CPU	2141.77ms	39.89ms (1.86%)	12.61ms (0.59%)	-1.08%

Table A.2: Compilation times of all solution programs of the problem Tablice kierunkowe (tab) from the finale of XXII Polish Olimpiad in Informatics. For each configuration (Solution program and Sandbox columns) the data was collected from 10 runs. Real and CPU times were collected from the same runs. Slowdown is measured from the times of configuration without the sandbox.

A.1.3. Modernizacja autostrady (mod)

Solution program	Sandbox	Time	Mean	Std. dev.	Std. err. on the mean	Slowdown
mod.cpp	no	real	506.86ms	8.18ms (1.61%)	2.59ms (0.51%)	0.00%
		CPU	494.46ms	7.64ms (1.54%)	2.42ms (0.49%)	0.00%
	yes	real	545.40ms	7.35ms (1.35%)	2.33ms (0.43%)	7.60%
		CPU	537.60ms	6.49ms (1.21%)	2.05ms (0.38%)	8.72%
	no BPF	real	453.21ms	45.80ms (10.11%)	14.48ms (3.20%)	-10.59%
		CPU	445.80ms	42.17ms (9.46%)	13.33ms (2.99%)	-9.84%
mod2.cpp	no	real	617.70ms	48.98ms (7.93%)	15.49ms (2.51%)	0.00%
		CPU	608.57ms	44.07ms (7.24%)	13.94ms (2.29%)	0.00%
	yes	real	699.67ms	15.03ms (2.15%)	4.75ms (0.68%)	13.27%
		CPU	688.35ms	11.55ms (1.68%)	3.65ms (0.53%)	13.11%
	no BPF	real	633.41ms	14.04ms (2.22%)	4.44ms (0.70%)	2.54%
		CPU	619.64ms	8.23ms (1.33%)	2.60ms (0.42%)	1.82%
mod3.cpp	no	real	640.60ms	9.78ms (1.53%)	3.09ms (0.48%)	0.00%
		CPU	634.93ms	9.92ms (1.56%)	3.14ms (0.49%)	0.00%
	yes	real	706.43ms	18.20ms (2.58%)	5.75ms (0.81%)	10.28%
		CPU	688.07ms	6.31ms (0.92%)	1.99ms (0.29%)	8.37%
	no BPF	real	641.73ms	15.47ms (2.41%)	4.89ms (0.76%)	0.18%
		CPU	627.42ms	10.60ms (1.69%)	3.35ms (0.53%)	-1.18%
mod4.cpp	no	real	2215.50ms	35.85ms (1.62%)	11.34ms (0.51%)	0.00%
		CPU	2195.65ms	34.97ms (1.59%)	11.06ms (0.50%)	0.00%
	yes	real	2420.52ms	147.40ms (6.09%)	46.61ms (1.93%)	9.25%
		CPU	2400.51ms	141.98ms (5.91%)	44.90ms (1.87%)	9.33%
	no BPF	real	2208.39ms	38.47ms (1.74%)	12.17ms (0.55%)	-0.32%
		CPU	2182.31ms	32.76ms (1.50%)	10.36ms (0.47%)	-0.61%
mod5.cpp	no	real	387.92ms	34.66ms (8.93%)	10.96ms (2.83%)	0.00%
		CPU	384.44ms	31.66ms (8.23%)	10.01ms (2.60%)	0.00%
	yes	real	478.65ms	21.47ms (4.49%)	6.79ms (1.42%)	23.39%
		CPU	468.80ms	22.54ms (4.81%)	7.13ms (1.52%)	21.94%
	no BPF	real	446.20ms	7.88ms (1.77%)	2.49ms (0.56%)	15.03%
		CPU	438.35ms	8.13ms (1.86%)	2.57ms (0.59%)	14.02%
modb1.cpp	no	real	645.45ms	14.49ms (2.25%)	4.58ms (0.71%)	0.00%
		CPU	639.14ms	11.80ms (1.85%)	3.73ms (0.58%)	0.00%
	yes	real	704.78ms	11.64ms (1.65%)	3.68ms (0.52%)	9.19%
		CPU	691.52ms	9.97ms (1.44%)	3.15ms (0.46%)	8.19%
	no BPF	real	572.02ms	58.27ms (10.19%)	18.43ms (3.22%)	-11.38%
		CPU	566.08ms	55.26ms (9.76%)	17.47ms (3.09%)	-11.43%
modb2.cpp	no	real	649.10ms	9.53ms (1.47%)	3.01ms (0.46%)	0.00%
		CPU	637.41ms	4.50ms (0.71%)	1.42ms (0.22%)	0.00%
	yes	real	700.70ms	10.35ms (1.48%)	3.27ms (0.47%)	7.95%
		CPU	689.42ms	6.08ms (0.88%)	1.92ms (0.28%)	8.16%
	no BPF	real	639.07ms	13.18ms (2.06%)	4.17ms (0.65%)	-1.55%
		CPU	626.76ms	10.97ms (1.75%)	3.47ms (0.55%)	-1.67%
modb3.cpp	no	real	657.89ms	12.21ms (1.86%)	3.86ms (0.59%)	0.00%
		CPU	648.33ms	12.27ms (1.89%)	3.88ms (0.60%)	0.00%
	yes	real	695.02ms	13.18ms (1.90%)	4.17ms (0.60%)	5.64%
		CPU	687.97ms	12.59ms (1.83%)	3.98ms (0.58%)	6.12%
	no BPF	real	633.97ms	12.16ms (1.92%)	3.85ms (0.61%)	-3.64%
		CPU	625.14ms	11.16ms (1.78%)	3.53ms (0.56%)	-3.58%

modb4.cpp	no	real	651.47ms	14.58ms (2.24%)	4.61ms (0.71%)	0.00%
		CPU	639.38ms	8.95ms (1.40%)	2.83ms (0.44%)	0.00%
	yes	real	702.43ms	14.35ms (2.04%)	4.54ms (0.65%)	7.82%
		CPU	690.45ms	10.40ms (1.51%)	3.29ms (0.48%)	7.99%
	no BPF	real	641.23ms	11.85ms (1.85%)	3.75ms (0.58%)	-1.57%
		CPU	628.03ms	8.47ms (1.35%)	2.68ms (0.43%)	-1.77%
modb5.cpp	no	real	648.21ms	9.03ms (1.39%)	2.86ms (0.44%)	0.00%
		CPU	638.99ms	9.92ms (1.55%)	3.14ms (0.49%)	0.00%
	yes	real	696.55ms	10.56ms (1.52%)	3.34ms (0.48%)	7.46%
		CPU	686.65ms	8.27ms (1.20%)	2.62ms (0.38%)	7.46%
	no BPF	real	631.47ms	10.46ms (1.66%)	3.31ms (0.52%)	-2.58%
		CPU	623.31ms	6.18ms (0.99%)	1.95ms (0.31%)	-2.45%
mods1.cpp	no	real	501.15ms	9.42ms (1.88%)	2.98ms (0.59%)	0.00%
		CPU	493.57ms	9.06ms (1.84%)	2.87ms (0.58%)	0.00%
	yes	real	549.66ms	9.73ms (1.77%)	3.08ms (0.56%)	9.68%
		CPU	537.71ms	8.86ms (1.65%)	2.80ms (0.52%)	8.94%
	no BPF	real	489.43ms	10.50ms (2.14%)	3.32ms (0.68%)	-2.34%
		CPU	480.21ms	7.46ms (1.55%)	2.36ms (0.49%)	-2.71%
mods2.cpp	no	real	493.88ms	9.76ms (1.98%)	3.09ms (0.63%)	0.00%
		CPU	484.96ms	6.33ms (1.31%)	2.00ms (0.41%)	0.00%
	yes	real	542.43ms	8.03ms (1.48%)	2.54ms (0.47%)	9.83%
		CPU	530.13ms	5.03ms (0.95%)	1.59ms (0.30%)	9.31%
	no BPF	real	482.53ms	7.30ms (1.51%)	2.31ms (0.48%)	-2.30%
		CPU	469.91ms	5.84ms (1.24%)	1.85ms (0.39%)	-3.10%
mods3.cpp	no	real	509.10ms	10.04ms (1.97%)	3.17ms (0.62%)	0.00%
		CPU	497.64ms	5.89ms (1.18%)	1.86ms (0.37%)	0.00%
	yes	real	546.91ms	5.99ms (1.09%)	1.89ms (0.35%)	7.43%
		CPU	540.42ms	4.48ms (0.83%)	1.42ms (0.26%)	8.60%
	no BPF	real	487.78ms	10.08ms (2.07%)	3.19ms (0.65%)	-4.19%
		CPU	482.33ms	10.09ms (2.09%)	3.19ms (0.66%)	-3.08%

Table A.3: Compilation times of all solution programs of the problem Modernizacja autostrady (mod) from the finale of XXII Polish Olimpiad in Informatics. For each configuration (Solution program and Sandbox columns) the data was collected from 10 runs. Real and CPU times were collected from the same runs. Slowdown is measured from the times of configuration without the sandbox.

A.1.4. Wycieczki (wyc)

Solution program	Sandbox	Time	Mean	Std. dev.	Std. err. on the mean	Slowdown
wyc.cpp	no	real	1020.39ms	96.63ms (9.47%)	30.56ms (2.99%)	0.00%
		CPU	1010.54ms	92.75ms (9.18%)	29.33ms (2.90%)	0.00%
	yes	real	1166.98ms	25.27ms (2.17%)	7.99ms (0.68%)	14.37%
		CPU	1147.29ms	23.21ms (2.02%)	7.34ms (0.64%)	13.53%
	no BPF	real	1062.37ms	17.42ms (1.64%)	5.51ms (0.52%)	4.11%
		CPU	1044.65ms	5.48ms (0.52%)	1.73ms (0.17%)	3.38%

wyc0.pas	no	real	70.80ms	3.11ms (4.39%)	0.98ms (1.39%)	0.00%
		CPU	69.32ms	3.27ms (4.72%)	1.03ms (1.49%)	0.00%
	yes	real	68.40ms	4.93ms (7.21%)	1.56ms (2.28%)	-3.39%
		CPU	65.60ms	4.60ms (7.01%)	1.46ms (2.22%)	-5.36%
	no BPF	real	66.51ms	3.06ms (4.61%)	0.97ms (1.46%)	-6.05%
		CPU	64.84ms	3.10ms (4.79%)	0.98ms (1.51%)	-6.46%
wyc2.cpp	no	real	320.31ms	25.17ms (7.86%)	7.96ms (2.48%)	0.00%
		CPU	313.19ms	21.55ms (6.88%)	6.82ms (2.18%)	0.00%
	yes	real	283.97ms	18.16ms (6.39%)	5.74ms (2.02%)	-11.34%
		CPU	281.69ms	18.26ms (6.48%)	5.77ms (2.05%)	-10.06%
	no BPF	real	321.54ms	9.08ms (2.82%)	2.87ms (0.89%)	0.39%
		CPU	311.98ms	6.04ms (1.94%)	1.91ms (0.61%)	-0.39%
wycb1.cpp	no	real	949.95ms	17.50ms (1.84%)	5.53ms (0.58%)	0.00%
		CPU	941.99ms	12.97ms (1.38%)	4.10ms (0.44%)	0.00%
	yes	real	1025.31ms	18.12ms (1.77%)	5.73ms (0.56%)	7.93%
		CPU	1012.82ms	11.24ms (1.11%)	3.55ms (0.35%)	7.52%
	no BPF	real	942.95ms	15.54ms (1.65%)	4.91ms (0.52%)	-0.74%
		CPU	924.64ms	12.31ms (1.33%)	3.89ms (0.42%)	-1.84%
wycb2.cpp	no	real	1073.49ms	9.98ms (0.93%)	3.16ms (0.29%)	0.00%
		CPU	1064.54ms	6.97ms (0.65%)	2.20ms (0.21%)	0.00%
	yes	real	1160.66ms	59.32ms (5.11%)	18.76ms (1.62%)	8.12%
		CPU	1145.10ms	56.74ms (4.95%)	17.94ms (1.57%)	7.57%
	no BPF	real	1031.55ms	102.81ms (9.97%)	32.51ms (3.15%)	-3.91%
		CPU	1019.14ms	97.90ms (9.61%)	30.96ms (3.04%)	-4.27%
wycb3.cpp	no	real	693.59ms	9.13ms (1.32%)	2.89ms (0.42%)	0.00%
		CPU	684.05ms	8.69ms (1.27%)	2.75ms (0.40%)	0.00%
	yes	real	752.42ms	8.78ms (1.17%)	2.78ms (0.37%)	8.48%
		CPU	739.72ms	6.32ms (0.85%)	2.00ms (0.27%)	8.14%
	no BPF	real	679.08ms	11.68ms (1.72%)	3.69ms (0.54%)	-2.09%
		CPU	668.80ms	8.52ms (1.27%)	2.69ms (0.40%)	-2.23%
wycb4.cpp	no	real	1097.15ms	18.26ms (1.66%)	5.77ms (0.53%)	0.00%
		CPU	1085.43ms	17.18ms (1.58%)	5.43ms (0.50%)	0.00%
	yes	real	1084.24ms	108.35ms (9.99%)	34.26ms (3.16%)	-1.18%
		CPU	1070.69ms	101.89ms (9.52%)	32.22ms (3.01%)	-1.36%
	no BPF	real	1074.07ms	21.18ms (1.97%)	6.70ms (0.62%)	-2.10%
		CPU	1060.03ms	18.00ms (1.70%)	5.69ms (0.54%)	-2.34%
wycb5.cpp	no	real	1014.67ms	24.17ms (2.38%)	7.64ms (0.75%)	0.00%
		CPU	997.87ms	11.21ms (1.12%)	3.54ms (0.36%)	0.00%
	yes	real	1111.70ms	19.76ms (1.78%)	6.25ms (0.56%)	9.56%
		CPU	1091.58ms	14.61ms (1.34%)	4.62ms (0.42%)	9.39%
	no BPF	real	996.27ms	14.96ms (1.50%)	4.73ms (0.47%)	-1.81%
		CPU	982.35ms	12.70ms (1.29%)	4.02ms (0.41%)	-1.56%

Table A.4: Compilation times of all solution programs of the problem Wycieczki (wyc) from the finale of XXII Polish Olimpiad in Informatics. For each configuration (Solution program and Sandbox columns) the data was collected from 10 runs. Real and CPU times were collected from the same runs. Slowdown is measured from the times of configuration without the sandbox.

A.2. Model solutions' run times

A.2.1. Myjnie (myj)

Test case	Sandbox	Time	Mean	Std. dev.	Std. err. on the mean	Slowdown
myj0	no	real	2.41ms	0.61ms (25.26%)	0.19ms (7.99%)	0.00%
		CPU	2.45ms	0.60ms (24.58%)	0.19ms (7.77%)	0.00%
	yes	real	2.18ms	0.44ms (20.27%)	0.14ms (6.41%)	-9.74%
		CPU	2.03ms	0.44ms (21.61%)	0.14ms (6.83%)	-17.02%
myj1ocen	no	real	1.95ms	0.35ms (17.65%)	0.11ms (5.58%)	0.00%
		CPU	2.00ms	0.35ms (17.38%)	0.11ms (5.50%)	0.00%
	yes	real	2.14ms	0.75ms (35.07%)	0.24ms (11.09%)	9.63%
		CPU	1.96ms	0.74ms (37.70%)	0.23ms (11.92%)	-1.68%
myj2ocen	no	real	0.91ms	0.04ms (4.71%)	0.01ms (1.49%)	0.00%
		CPU	0.98ms	0.05ms (5.06%)	0.02ms (1.60%)	0.00%
	yes	real	0.97ms	0.40ms (40.86%)	0.13ms (12.92%)	6.62%
		CPU	0.87ms	0.39ms (44.91%)	0.12ms (14.20%)	-11.92%
myj3ocen	no	real	337.52ms	7.81ms (2.31%)	2.47ms (0.73%)	0.00%
		CPU	330.51ms	2.65ms (0.80%)	0.84ms (0.25%)	0.00%
	yes	real	266.05ms	10.76ms (4.04%)	3.40ms (1.28%)	-21.17%
		CPU	264.19ms	7.94ms (3.01%)	2.51ms (0.95%)	-20.06%
myj1a	no	real	1.01ms	0.55ms (54.48%)	0.17ms (17.23%)	0.00%
		CPU	0.89ms	0.01ms (1.39%)	0.00ms (0.44%)	0.00%
	yes	real	0.73ms	0.03ms (4.07%)	0.01ms (1.29%)	-27.65%
		CPU	0.63ms	0.01ms (1.27%)	0.00ms (0.40%)	-29.17%
myj1b	no	real	0.85ms	0.03ms (3.86%)	0.01ms (1.22%)	0.00%
		CPU	0.90ms	0.02ms (2.30%)	0.01ms (0.73%)	0.00%
	yes	real	0.72ms	0.03ms (4.14%)	0.01ms (1.31%)	-14.44%
		CPU	0.63ms	0.01ms (2.28%)	0.00ms (0.72%)	-30.26%
myj1c	no	real	0.85ms	0.04ms (5.22%)	0.01ms (1.65%)	0.00%
		CPU	0.91ms	0.03ms (3.39%)	0.01ms (1.07%)	0.00%
	yes	real	0.73ms	0.03ms (3.91%)	0.01ms (1.24%)	-14.10%
		CPU	0.64ms	0.01ms (2.09%)	0.00ms (0.66%)	-29.62%
myj1d	no	real	2.13ms	0.55ms (25.63%)	0.17ms (8.10%)	0.00%
		CPU	2.16ms	0.56ms (25.76%)	0.18ms (8.15%)	0.00%
	yes	real	1.69ms	0.06ms (3.62%)	0.02ms (1.14%)	-20.51%
		CPU	1.55ms	0.05ms (2.97%)	0.01ms (0.94%)	-28.28%
myj1e	no	real	1.62ms	0.42ms (26.09%)	0.13ms (8.25%)	0.00%
		CPU	1.66ms	0.39ms (23.43%)	0.12ms (7.41%)	0.00%
	yes	real	1.35ms	0.28ms (21.12%)	0.09ms (6.68%)	-16.64%
		CPU	1.22ms	0.28ms (22.98%)	0.09ms (7.27%)	-26.08%
myj1f	no	real	1.87ms	0.30ms (16.09%)	0.09ms (5.09%)	0.00%
		CPU	1.92ms	0.30ms (15.41%)	0.09ms (4.87%)	0.00%
	yes	real	1.96ms	0.66ms (33.54%)	0.21ms (10.61%)	5.06%
		CPU	1.82ms	0.63ms (34.83%)	0.20ms (11.01%)	-5.16%
myj1g	no	real	1.78ms	0.05ms (3.04%)	0.02ms (0.96%)	0.00%
		CPU	1.82ms	0.04ms (2.09%)	0.01ms (0.66%)	0.00%
	yes	real	1.99ms	0.69ms (34.57%)	0.22ms (10.93%)	11.80%
		CPU	1.82ms	0.63ms (34.46%)	0.20ms (10.90%)	0.14%
myj1h	no	real	2.03ms	0.44ms (21.53%)	0.14ms (6.81%)	0.00%
		CPU	1.94ms	0.29ms (15.12%)	0.09ms (4.78%)	0.00%
	yes	real	2.05ms	0.68ms (33.37%)	0.22ms (10.55%)	1.13%
		CPU	1.72ms	0.38ms (22.04%)	0.12ms (6.97%)	-11.76%

myj1i	no	real	2.35ms	0.77ms (32.77%)	0.24ms (10.36%)	0.00%
		CPU	2.38ms	0.74ms (30.95%)	0.23ms (9.79%)	0.00%
	yes	real	2.21ms	0.83ms (37.43%)	0.26ms (11.84%)	-6.06%
		CPU	2.07ms	0.81ms (38.93%)	0.26ms (12.31%)	-12.92%
myj1j	no	real	1.59ms	0.60ms (37.47%)	0.19ms (11.85%)	0.00%
		CPU	1.65ms	0.59ms (35.87%)	0.19ms (11.34%)	0.00%
	yes	real	1.74ms	0.62ms (35.38%)	0.19ms (11.19%)	9.20%
		CPU	1.57ms	0.62ms (39.83%)	0.20ms (12.60%)	-5.07%
myj2a	no	real	2.30ms	0.07ms (3.02%)	0.02ms (0.95%)	0.00%
		CPU	2.32ms	0.04ms (1.54%)	0.01ms (0.49%)	0.00%
	yes	real	2.75ms	0.76ms (27.81%)	0.24ms (8.79%)	19.38%
		CPU	2.45ms	0.68ms (27.72%)	0.22ms (8.77%)	5.75%
myj2b	no	real	2.56ms	0.76ms (29.56%)	0.24ms (9.35%)	0.00%
		CPU	2.41ms	0.31ms (12.68%)	0.10ms (4.01%)	0.00%
	yes	real	2.34ms	0.32ms (13.56%)	0.10ms (4.29%)	-8.62%
		CPU	2.18ms	0.29ms (13.38%)	0.09ms (4.23%)	-9.52%
myj2c	no	real	2.69ms	0.57ms (21.11%)	0.18ms (6.67%)	0.00%
		CPU	2.68ms	0.58ms (21.47%)	0.18ms (6.79%)	0.00%
	yes	real	3.14ms	0.85ms (27.02%)	0.27ms (8.55%)	16.66%
		CPU	2.94ms	0.84ms (28.66%)	0.27ms (9.06%)	9.51%
myj3a	no	real	4.09ms	0.85ms (20.84%)	0.27ms (6.59%)	0.00%
		CPU	4.10ms	0.84ms (20.59%)	0.27ms (6.51%)	0.00%
	yes	real	3.60ms	0.69ms (19.12%)	0.22ms (6.05%)	-11.91%
		CPU	3.43ms	0.64ms (18.64%)	0.20ms (5.89%)	-16.18%
myj3b	no	real	3.53ms	0.66ms (18.58%)	0.21ms (5.88%)	0.00%
		CPU	3.51ms	0.60ms (17.00%)	0.19ms (5.38%)	0.00%
	yes	real	3.78ms	0.79ms (21.00%)	0.25ms (6.64%)	7.05%
		CPU	3.61ms	0.80ms (22.19%)	0.25ms (7.02%)	2.83%
myj4a	no	real	6.04ms	0.94ms (15.52%)	0.30ms (4.91%)	0.00%
		CPU	5.74ms	0.78ms (13.65%)	0.25ms (4.32%)	0.00%
	yes	real	5.68ms	0.69ms (12.20%)	0.22ms (3.86%)	-5.86%
		CPU	5.49ms	0.68ms (12.35%)	0.21ms (3.90%)	-4.41%
myj4b	no	real	4.06ms	0.77ms (18.94%)	0.24ms (5.99%)	0.00%
		CPU	3.77ms	0.43ms (11.34%)	0.14ms (3.59%)	0.00%
	yes	real	4.50ms	0.91ms (20.20%)	0.29ms (6.39%)	10.63%
		CPU	4.14ms	0.83ms (20.01%)	0.26ms (6.33%)	9.92%
myj5	no	real	5.10ms	0.90ms (17.64%)	0.28ms (5.58%)	0.00%
		CPU	4.89ms	0.80ms (16.43%)	0.25ms (5.20%)	0.00%
	yes	real	4.57ms	0.98ms (21.43%)	0.31ms (6.78%)	-10.49%
		CPU	4.31ms	0.86ms (20.00%)	0.27ms (6.32%)	-11.88%
myj6	no	real	6.87ms	0.94ms (13.71%)	0.30ms (4.33%)	0.00%
		CPU	6.63ms	0.75ms (11.34%)	0.24ms (3.59%)	0.00%
	yes	real	6.51ms	1.20ms (18.38%)	0.38ms (5.81%)	-5.22%
		CPU	6.05ms	0.88ms (14.59%)	0.28ms (4.61%)	-8.78%
myj7	no	real	7.69ms	1.47ms (19.06%)	0.46ms (6.03%)	0.00%
		CPU	7.15ms	0.75ms (10.45%)	0.24ms (3.30%)	0.00%
	yes	real	6.87ms	0.90ms (13.11%)	0.29ms (4.15%)	-10.62%
		CPU	6.60ms	0.86ms (13.06%)	0.27ms (4.13%)	-7.67%
myj8	no	real	11.73ms	0.86ms (7.34%)	0.27ms (2.32%)	0.00%
		CPU	11.51ms	1.14ms (9.90%)	0.36ms (3.13%)	0.00%
	yes	real	10.89ms	1.31ms (12.05%)	0.42ms (3.81%)	-7.14%
		CPU	10.53ms	1.20ms (11.41%)	0.38ms (3.61%)	-8.52%

myj9	no	real	10.82ms	1.34ms (12.36%)	0.42ms (3.91%)	0.00%
		CPU	10.14ms	0.84ms (8.32%)	0.27ms (2.63%)	0.00%
	yes	real	9.60ms	1.15ms (11.96%)	0.36ms (3.78%)	-11.21%
		CPU	9.39ms	1.17ms (12.42%)	0.37ms (3.93%)	-7.43%
myj10	no	real	18.74ms	1.59ms (8.48%)	0.50ms (2.68%)	0.00%
		CPU	17.82ms	1.28ms (7.18%)	0.40ms (2.27%)	0.00%
	yes	real	17.20ms	1.65ms (9.57%)	0.52ms (3.03%)	-8.23%
		CPU	16.94ms	1.64ms (9.67%)	0.52ms (3.06%)	-4.94%
myj11a	no	real	10.95ms	1.25ms (11.45%)	0.40ms (3.62%)	0.00%
		CPU	10.72ms	0.78ms (7.25%)	0.25ms (2.29%)	0.00%
	yes	real	10.22ms	0.20ms (1.94%)	0.06ms (0.61%)	-6.71%
		CPU	9.97ms	0.19ms (1.91%)	0.06ms (0.60%)	-6.97%
myj11b	no	real	10.68ms	0.44ms (4.14%)	0.14ms (1.31%)	0.00%
		CPU	10.45ms	0.19ms (1.86%)	0.06ms (0.59%)	0.00%
	yes	real	10.29ms	0.24ms (2.34%)	0.08ms (0.74%)	-3.67%
		CPU	10.05ms	0.19ms (1.91%)	0.06ms (0.60%)	-3.89%
myj11c	no	real	10.27ms	0.43ms (4.24%)	0.14ms (1.34%)	0.00%
		CPU	10.02ms	0.23ms (2.31%)	0.07ms (0.73%)	0.00%
	yes	real	9.78ms	0.13ms (1.36%)	0.04ms (0.43%)	-4.72%
		CPU	9.53ms	0.15ms (1.55%)	0.05ms (0.49%)	-4.95%
myj12a	no	real	19.54ms	0.45ms (2.33%)	0.14ms (0.74%)	0.00%
		CPU	19.28ms	0.18ms (0.93%)	0.06ms (0.30%)	0.00%
	yes	real	18.90ms	0.13ms (0.70%)	0.04ms (0.22%)	-3.24%
		CPU	18.63ms	0.14ms (0.74%)	0.04ms (0.23%)	-3.36%
myj12b	no	real	19.46ms	0.65ms (3.33%)	0.20ms (1.05%)	0.00%
		CPU	19.19ms	0.52ms (2.69%)	0.16ms (0.85%)	0.00%
	yes	real	18.92ms	0.15ms (0.78%)	0.05ms (0.25%)	-2.76%
		CPU	18.63ms	0.15ms (0.80%)	0.05ms (0.25%)	-2.90%
myj12c	no	real	19.62ms	0.43ms (2.20%)	0.14ms (0.69%)	0.00%
		CPU	19.36ms	0.33ms (1.69%)	0.10ms (0.53%)	0.00%
	yes	real	22.64ms	2.75ms (12.13%)	0.87ms (3.84%)	15.35%
		CPU	22.08ms	2.87ms (13.02%)	0.91ms (4.12%)	14.03%
myj12d	no	real	22.57ms	1.70ms (7.52%)	0.54ms (2.38%)	0.00%
		CPU	21.98ms	1.87ms (8.53%)	0.59ms (2.70%)	0.00%
	yes	real	22.75ms	2.47ms (10.86%)	0.78ms (3.43%)	0.78%
		CPU	22.17ms	2.23ms (10.08%)	0.71ms (3.19%)	0.82%
myj13	no	real	22.06ms	2.24ms (10.16%)	0.71ms (3.21%)	0.00%
		CPU	21.17ms	1.68ms (7.91%)	0.53ms (2.50%)	0.00%
	yes	real	22.28ms	3.41ms (15.31%)	1.08ms (4.84%)	0.98%
		CPU	21.05ms	2.43ms (11.53%)	0.77ms (3.65%)	-0.55%
myj14	no	real	110.00ms	6.34ms (5.76%)	2.01ms (1.82%)	0.00%
		CPU	107.97ms	4.62ms (4.27%)	1.46ms (1.35%)	0.00%
	yes	real	108.22ms	6.18ms (5.71%)	1.95ms (1.81%)	-1.62%
		CPU	107.58ms	6.15ms (5.72%)	1.94ms (1.81%)	-0.36%
myj15a	no	real	40.84ms	4.07ms (9.96%)	1.29ms (3.15%)	0.00%
		CPU	39.52ms	3.42ms (8.65%)	1.08ms (2.73%)	0.00%
	yes	real	40.53ms	2.82ms (6.97%)	0.89ms (2.20%)	-0.75%
		CPU	39.79ms	2.46ms (6.18%)	0.78ms (1.96%)	0.69%
myj15b	no	real	40.66ms	4.27ms (10.51%)	1.35ms (3.32%)	0.00%
		CPU	39.79ms	3.76ms (9.45%)	1.19ms (2.99%)	0.00%
	yes	real	39.83ms	3.40ms (8.52%)	1.07ms (2.70%)	-2.04%
		CPU	38.65ms	3.08ms (7.96%)	0.97ms (2.52%)	-2.86%

myj15c	no	real	40.38ms	3.03ms (7.51%)	0.96ms (2.37%)	0.00%
		CPU	39.78ms	2.86ms (7.20%)	0.91ms (2.28%)	0.00%
	yes	real	40.14ms	2.51ms (6.25%)	0.79ms (1.98%)	-0.60%
		CPU	39.54ms	2.36ms (5.97%)	0.75ms (1.89%)	-0.60%
myj16a	no	real	420.55ms	8.64ms (2.05%)	2.73ms (0.65%)	0.00%
		CPU	412.12ms	3.04ms (0.74%)	0.96ms (0.23%)	0.00%
	yes	real	367.88ms	38.53ms (10.47%)	12.19ms (3.31%)	-12.52%
		CPU	365.03ms	36.65ms (10.04%)	11.59ms (3.18%)	-11.43%
myj16b	no	real	389.42ms	40.58ms (10.42%)	12.83ms (3.30%)	0.00%
		CPU	381.49ms	35.42ms (9.29%)	11.20ms (2.94%)	0.00%
	yes	real	411.51ms	2.41ms (0.59%)	0.76ms (0.19%)	5.67%
		CPU	409.79ms	2.41ms (0.59%)	0.76ms (0.19%)	7.42%
myj16c	no	real	381.73ms	6.34ms (1.66%)	2.00ms (0.52%)	0.00%
		CPU	376.18ms	1.92ms (0.51%)	0.61ms (0.16%)	0.00%
	yes	real	386.79ms	7.18ms (1.86%)	2.27ms (0.59%)	1.33%
		CPU	378.28ms	5.92ms (1.57%)	1.87ms (0.49%)	0.56%
myj17a	no	real	77.59ms	5.07ms (6.53%)	1.60ms (2.06%)	0.00%
		CPU	77.16ms	4.84ms (6.27%)	1.53ms (1.98%)	0.00%
	yes	real	77.15ms	4.62ms (5.99%)	1.46ms (1.89%)	-0.56%
		CPU	76.61ms	4.63ms (6.04%)	1.46ms (1.91%)	-0.72%
myj17b	no	real	78.77ms	5.20ms (6.60%)	1.64ms (2.09%)	0.00%
		CPU	76.09ms	4.07ms (5.35%)	1.29ms (1.69%)	0.00%
	yes	real	78.29ms	4.46ms (5.69%)	1.41ms (1.80%)	-0.61%
		CPU	76.57ms	4.43ms (5.78%)	1.40ms (1.83%)	0.64%
myj17c	no	real	79.24ms	6.06ms (7.65%)	1.92ms (2.42%)	0.00%
		CPU	77.95ms	5.76ms (7.39%)	1.82ms (2.34%)	0.00%
	yes	real	78.64ms	5.14ms (6.54%)	1.63ms (2.07%)	-0.75%
		CPU	77.20ms	3.70ms (4.79%)	1.17ms (1.51%)	-0.95%
myj18a	no	real	43.72ms	3.24ms (7.42%)	1.03ms (2.35%)	0.00%
		CPU	42.78ms	2.76ms (6.44%)	0.87ms (2.04%)	0.00%
	yes	real	42.89ms	4.46ms (10.39%)	1.41ms (3.29%)	-1.91%
		CPU	41.88ms	3.99ms (9.53%)	1.26ms (3.01%)	-2.12%
myj18b	no	real	45.58ms	3.73ms (8.19%)	1.18ms (2.59%)	0.00%
		CPU	43.68ms	3.15ms (7.22%)	1.00ms (2.28%)	0.00%
	yes	real	43.86ms	3.62ms (8.25%)	1.14ms (2.61%)	-3.76%
		CPU	43.42ms	3.62ms (8.33%)	1.14ms (2.63%)	-0.61%
myj18c	no	real	42.52ms	4.86ms (11.42%)	1.54ms (3.61%)	0.00%
		CPU	39.70ms	2.63ms (6.64%)	0.83ms (2.10%)	0.00%
	yes	real	42.83ms	4.52ms (10.56%)	1.43ms (3.34%)	0.73%
		CPU	41.58ms	4.36ms (10.48%)	1.38ms (3.31%)	4.74%
myj19a	no	real	1353.52ms	16.71ms (1.23%)	5.28ms (0.39%)	0.00%
		CPU	1346.57ms	11.16ms (0.83%)	3.53ms (0.26%)	0.00%
	yes	real	1269.57ms	119.68ms (9.43%)	37.85ms (2.98%)	-6.20%
		CPU	1258.78ms	116.01ms (9.22%)	36.69ms (2.91%)	-6.52%
myj19b	no	real	1350.30ms	14.62ms (1.08%)	4.62ms (0.34%)	0.00%
		CPU	1342.81ms	11.45ms (0.85%)	3.62ms (0.27%)	0.00%
	yes	real	1376.87ms	17.28ms (1.26%)	5.47ms (0.40%)	1.97%
		CPU	1361.73ms	8.43ms (0.62%)	2.67ms (0.20%)	1.41%
myj20a	no	real	1283.10ms	113.29ms (8.83%)	35.83ms (2.79%)	0.00%
		CPU	1277.37ms	110.65ms (8.66%)	34.99ms (2.74%)	0.00%
	yes	real	1278.21ms	120.55ms (9.43%)	38.12ms (2.98%)	-0.38%
		CPU	1269.97ms	123.81ms (9.75%)	39.15ms (3.08%)	-0.58%

myj20b	no	real	1211.57ms	111.11ms (9.17%)	35.14ms (2.90%)	0.00%
		CPU	1203.19ms	107.88ms (8.97%)	34.11ms (2.84%)	0.00%
	yes	real	1366.10ms	18.91ms (1.38%)	5.98ms (0.44%)	12.75%
		CPU	1350.41ms	13.59ms (1.01%)	4.30ms (0.32%)	12.24%
myj20c	no	real	1354.07ms	15.61ms (1.15%)	4.94ms (0.36%)	0.00%
		CPU	1349.44ms	13.83ms (1.02%)	4.37ms (0.32%)	0.00%
	yes	real	1352.38ms	20.36ms (1.51%)	6.44ms (0.48%)	-0.12%
		CPU	1341.98ms	13.65ms (1.02%)	4.32ms (0.32%)	-0.55%
myj20d	no	real	1106.04ms	18.56ms (1.68%)	5.87ms (0.53%)	0.00%
		CPU	1100.80ms	12.40ms (1.13%)	3.92ms (0.36%)	0.00%
	yes	real	1320.44ms	77.02ms (5.83%)	24.36ms (1.84%)	19.38%
		CPU	1313.12ms	77.91ms (5.93%)	24.64ms (1.88%)	19.29%
myj20e	no	real	1269.13ms	115.99ms (9.14%)	36.68ms (2.89%)	0.00%
		CPU	1263.02ms	115.20ms (9.12%)	36.43ms (2.88%)	0.00%
	yes	real	1364.85ms	19.99ms (1.46%)	6.32ms (0.46%)	7.54%
		CPU	1357.45ms	16.52ms (1.22%)	5.22ms (0.38%)	7.48%

Table A.5: Run times of the model solution program of the problem Myjnie (myj) from the finale of XXII Polish Olimpiad in Informatics. For each configuration (Test case and Sandbox columns) the data was collected from 10 runs. Real and CPU times were collected from the same runs. Slowdown is measured from the times of configuration without the sandbox.

A.2.2. Tablice kierunkowe (tab)

Test case	Sandbox	Time	Mean	Std. dev.	Std. err. on the mean	Slowdown
tab0	no	real	1.26ms	0.33ms (25.85%)	0.10ms (8.17%)	0.00%
		CPU	1.26ms	0.21ms (16.87%)	0.07ms (5.34%)	0.00%
	yes	real	0.72ms	0.04ms (5.35%)	0.01ms (1.69%)	-43.24%
		CPU	0.64ms	0.02ms (2.85%)	0.01ms (0.90%)	-49.35%
tab1ocen	no	real	1.69ms	0.77ms (45.56%)	0.24ms (14.41%)	0.00%
		CPU	1.55ms	0.58ms (37.26%)	0.18ms (11.78%)	0.00%
	yes	real	0.70ms	0.01ms (1.61%)	0.00ms (0.51%)	-58.46%
		CPU	0.63ms	0.01ms (1.90%)	0.00ms (0.60%)	-59.42%
tab2ocen	no	real	1.52ms	0.64ms (42.36%)	0.20ms (13.39%)	0.00%
		CPU	1.46ms	0.39ms (26.83%)	0.12ms (8.49%)	0.00%
	yes	real	0.81ms	0.32ms (39.39%)	0.10ms (12.46%)	-46.34%
		CPU	0.73ms	0.31ms (42.13%)	0.10ms (13.32%)	-50.25%
tab3ocen	no	real	55.10ms	3.93ms (7.13%)	1.24ms (2.25%)	0.00%
		CPU	54.22ms	2.05ms (3.79%)	0.65ms (1.20%)	0.00%
	yes	real	49.45ms	0.40ms (0.82%)	0.13ms (0.26%)	-10.26%
		CPU	49.16ms	0.41ms (0.84%)	0.13ms (0.27%)	-9.32%
tab1a	no	real	1.95ms	0.02ms (1.26%)	0.01ms (0.40%)	0.00%
		CPU	2.01ms	0.02ms (0.87%)	0.01ms (0.27%)	0.00%
	yes	real	1.22ms	0.02ms (1.96%)	0.01ms (0.62%)	-37.68%
		CPU	1.13ms	0.02ms (1.40%)	0.00ms (0.44%)	-43.91%
tab1b	no	real	1.56ms	0.02ms (1.17%)	0.01ms (0.37%)	0.00%
		CPU	1.62ms	0.01ms (0.91%)	0.00ms (0.29%)	0.00%
	yes	real	0.92ms	0.04ms (4.18%)	0.01ms (1.32%)	-41.12%
		CPU	0.83ms	0.03ms (3.14%)	0.01ms (0.99%)	-48.75%

tab1c	no	real	1.78ms	0.11ms (6.37%)	0.04ms (2.01%)	0.00%
		CPU	1.83ms	0.13ms (6.93%)	0.04ms (2.19%)	0.00%
	yes	real	1.09ms	0.02ms (1.67%)	0.01ms (0.53%)	-39.01%
		CPU	1.00ms	0.01ms (0.99%)	0.00ms (0.31%)	-45.67%
tab1d	no	real	1.62ms	0.01ms (0.57%)	0.00ms (0.18%)	0.00%
		CPU	1.68ms	0.01ms (0.39%)	0.00ms (0.12%)	0.00%
	yes	real	0.93ms	0.02ms (2.23%)	0.01ms (0.70%)	-42.46%
		CPU	0.85ms	0.01ms (1.34%)	0.00ms (0.42%)	-49.35%
tab1e	no	real	1.71ms	0.05ms (3.16%)	0.02ms (1.00%)	0.00%
		CPU	1.78ms	0.05ms (3.01%)	0.02ms (0.95%)	0.00%
	yes	real	1.07ms	0.02ms (2.28%)	0.01ms (0.72%)	-37.30%
		CPU	0.98ms	0.01ms (1.02%)	0.00ms (0.32%)	-44.79%
tab1f	no	real	1.87ms	0.05ms (2.82%)	0.02ms (0.89%)	0.00%
		CPU	1.94ms	0.06ms (2.88%)	0.02ms (0.91%)	0.00%
	yes	real	1.17ms	0.03ms (2.46%)	0.01ms (0.78%)	-37.60%
		CPU	1.07ms	0.02ms (1.74%)	0.01ms (0.55%)	-44.57%
tab1g	no	real	1.09ms	0.01ms (0.61%)	0.00ms (0.19%)	0.00%
		CPU	1.16ms	0.01ms (1.10%)	0.00ms (0.35%)	0.00%
	yes	real	0.71ms	0.02ms (3.03%)	0.01ms (0.96%)	-34.87%
		CPU	0.62ms	0.02ms (2.83%)	0.01ms (0.89%)	-46.27%
tab1h	no	real	1.11ms	0.01ms (1.03%)	0.00ms (0.33%)	0.00%
		CPU	1.18ms	0.01ms (0.85%)	0.00ms (0.27%)	0.00%
	yes	real	0.85ms	0.34ms (40.25%)	0.11ms (12.73%)	-23.74%
		CPU	0.75ms	0.31ms (41.76%)	0.10ms (13.21%)	-36.45%
tab1i	no	real	1.38ms	0.31ms (22.40%)	0.10ms (7.08%)	0.00%
		CPU	1.40ms	0.30ms (21.20%)	0.09ms (6.70%)	0.00%
	yes	real	0.90ms	0.35ms (39.25%)	0.11ms (12.41%)	-34.89%
		CPU	0.69ms	0.05ms (6.66%)	0.01ms (2.11%)	-50.55%
tab1j	no	real	1.27ms	0.34ms (27.09%)	0.11ms (8.57%)	0.00%
		CPU	1.31ms	0.30ms (23.05%)	0.10ms (7.29%)	0.00%
	yes	real	0.99ms	0.62ms (62.62%)	0.20ms (19.80%)	-21.98%
		CPU	0.91ms	0.61ms (67.17%)	0.19ms (21.24%)	-30.55%
tab1k	no	real	1.64ms	0.68ms (41.38%)	0.21ms (13.08%)	0.00%
		CPU	1.62ms	0.59ms (36.24%)	0.19ms (11.46%)	0.00%
	yes	real	0.93ms	0.37ms (39.39%)	0.12ms (12.46%)	-43.03%
		CPU	0.84ms	0.37ms (43.61%)	0.12ms (13.79%)	-48.02%
tab2a	no	real	3.77ms	0.78ms (20.80%)	0.25ms (6.58%)	0.00%
		CPU	3.44ms	0.04ms (1.29%)	0.01ms (0.41%)	0.00%
	yes	real	2.95ms	0.83ms (28.22%)	0.26ms (8.92%)	-21.55%
		CPU	2.83ms	0.79ms (28.02%)	0.25ms (8.86%)	-17.84%
tab2b	no	real	3.47ms	0.90ms (25.81%)	0.28ms (8.16%)	0.00%
		CPU	3.20ms	0.50ms (15.52%)	0.16ms (4.91%)	0.00%
	yes	real	1.61ms	0.32ms (20.09%)	0.10ms (6.35%)	-53.77%
		CPU	1.51ms	0.31ms (20.80%)	0.10ms (6.58%)	-52.82%
tab2c	no	real	2.07ms	0.03ms (1.61%)	0.01ms (0.51%)	0.00%
		CPU	2.12ms	0.03ms (1.49%)	0.01ms (0.47%)	0.00%
	yes	real	2.18ms	0.95ms (43.55%)	0.30ms (13.77%)	5.44%
		CPU	1.62ms	0.45ms (27.74%)	0.14ms (8.77%)	-23.78%
tab2d	no	real	3.24ms	0.59ms (18.24%)	0.19ms (5.77%)	0.00%
		CPU	3.32ms	0.61ms (18.32%)	0.19ms (5.79%)	0.00%
	yes	real	1.49ms	0.04ms (2.73%)	0.01ms (0.86%)	-53.80%
		CPU	1.41ms	0.03ms (2.00%)	0.01ms (0.63%)	-57.53%

tab2e	no	real	4.81ms	0.57ms (11.84%)	0.18ms (3.74%)	0.00%
		CPU	4.75ms	0.42ms (8.86%)	0.13ms (2.80%)	0.00%
	yes	real	3.49ms	0.83ms (23.68%)	0.26ms (7.49%)	-27.49%
		CPU	3.37ms	0.82ms (24.39%)	0.26ms (7.71%)	-29.04%
tab2f	no	real	3.33ms	0.51ms (15.23%)	0.16ms (4.82%)	0.00%
		CPU	3.37ms	0.51ms (14.99%)	0.16ms (4.74%)	0.00%
	yes	real	2.85ms	0.89ms (31.07%)	0.28ms (9.83%)	-14.44%
		CPU	2.33ms	0.66ms (28.39%)	0.21ms (8.98%)	-30.83%
tab3a	no	real	5.03ms	0.75ms (14.87%)	0.24ms (4.70%)	0.00%
		CPU	5.15ms	0.75ms (14.52%)	0.24ms (4.59%)	0.00%
	yes	real	4.04ms	0.80ms (19.77%)	0.25ms (6.25%)	-19.71%
		CPU	3.92ms	0.77ms (19.70%)	0.24ms (6.23%)	-23.91%
tab3b	no	real	4.08ms	0.82ms (20.12%)	0.26ms (6.36%)	0.00%
		CPU	3.90ms	0.73ms (18.64%)	0.23ms (5.89%)	0.00%
	yes	real	2.18ms	0.52ms (23.95%)	0.17ms (7.57%)	-46.64%
		CPU	2.08ms	0.50ms (24.12%)	0.16ms (7.63%)	-46.65%
tab3c	no	real	5.48ms	0.97ms (17.75%)	0.31ms (5.61%)	0.00%
		CPU	5.26ms	0.66ms (12.52%)	0.21ms (3.96%)	0.00%
	yes	real	3.55ms	0.79ms (22.30%)	0.25ms (7.05%)	-35.10%
		CPU	3.41ms	0.80ms (23.45%)	0.25ms (7.42%)	-35.24%
tab3d	no	real	4.13ms	0.92ms (22.21%)	0.29ms (7.02%)	0.00%
		CPU	3.84ms	0.47ms (12.35%)	0.15ms (3.91%)	0.00%
	yes	real	2.54ms	0.96ms (37.97%)	0.30ms (12.01%)	-38.58%
		CPU	2.42ms	0.91ms (37.71%)	0.29ms (11.93%)	-36.93%
tab3e	no	real	6.99ms	0.78ms (11.13%)	0.25ms (3.52%)	0.00%
		CPU	6.94ms	0.68ms (9.85%)	0.22ms (3.11%)	0.00%
	yes	real	5.36ms	0.79ms (14.74%)	0.25ms (4.66%)	-23.25%
		CPU	5.21ms	0.77ms (14.84%)	0.24ms (4.69%)	-25.00%
tab3f	no	real	3.84ms	0.51ms (13.17%)	0.16ms (4.16%)	0.00%
		CPU	3.88ms	0.50ms (12.92%)	0.16ms (4.09%)	0.00%
	yes	real	2.13ms	0.42ms (19.71%)	0.13ms (6.23%)	-44.55%
		CPU	2.01ms	0.39ms (19.48%)	0.12ms (6.16%)	-48.34%
tab4a	no	real	30.95ms	1.77ms (5.73%)	0.56ms (1.81%)	0.00%
		CPU	30.06ms	1.85ms (6.15%)	0.58ms (1.94%)	0.00%
	yes	real	27.12ms	2.29ms (8.46%)	0.73ms (2.68%)	-12.39%
		CPU	26.87ms	2.28ms (8.47%)	0.72ms (2.68%)	-10.60%
tab4b	no	real	10.63ms	1.54ms (14.45%)	0.49ms (4.57%)	0.00%
		CPU	10.43ms	1.60ms (15.36%)	0.51ms (4.86%)	0.00%
	yes	real	6.00ms	0.05ms (0.76%)	0.01ms (0.24%)	-43.53%
		CPU	5.87ms	0.02ms (0.36%)	0.01ms (0.11%)	-43.74%
tab4c	no	real	14.81ms	0.57ms (3.87%)	0.18ms (1.23%)	0.00%
		CPU	14.80ms	0.57ms (3.86%)	0.18ms (1.22%)	0.00%
	yes	real	10.46ms	0.30ms (2.87%)	0.09ms (0.91%)	-29.39%
		CPU	10.29ms	0.32ms (3.06%)	0.10ms (0.97%)	-30.45%
tab4d	no	real	11.03ms	0.36ms (3.28%)	0.11ms (1.04%)	0.00%
		CPU	11.03ms	0.36ms (3.25%)	0.11ms (1.03%)	0.00%
	yes	real	6.19ms	0.04ms (0.70%)	0.01ms (0.22%)	-43.84%
		CPU	6.05ms	0.02ms (0.30%)	0.01ms (0.10%)	-45.14%
tab4e	no	real	23.59ms	0.46ms (1.93%)	0.14ms (0.61%)	0.00%
		CPU	23.56ms	0.45ms (1.93%)	0.14ms (0.61%)	0.00%
	yes	real	19.39ms	0.69ms (3.58%)	0.22ms (1.13%)	-17.81%
		CPU	19.18ms	0.69ms (3.60%)	0.22ms (1.14%)	-18.60%

tab4f	no	real	10.19ms	1.16ms (11.37%)	0.37ms (3.60%)	0.00%
		CPU	10.20ms	1.15ms (11.29%)	0.36ms (3.57%)	0.00%
	yes	real	5.64ms	0.05ms (0.89%)	0.02ms (0.28%)	-44.67%
		CPU	5.49ms	0.02ms (0.41%)	0.01ms (0.13%)	-46.11%
tab5a	no	real	61.04ms	2.46ms (4.02%)	0.78ms (1.27%)	0.00%
		CPU	60.49ms	2.72ms (4.49%)	0.86ms (1.42%)	0.00%
	yes	real	56.89ms	1.85ms (3.25%)	0.59ms (1.03%)	-6.79%
		CPU	56.34ms	1.90ms (3.37%)	0.60ms (1.07%)	-6.85%
tab5b	no	real	17.09ms	1.29ms (7.57%)	0.41ms (2.39%)	0.00%
		CPU	16.72ms	1.30ms (7.76%)	0.41ms (2.45%)	0.00%
	yes	real	11.54ms	1.02ms (8.84%)	0.32ms (2.79%)	-32.47%
		CPU	11.38ms	0.99ms (8.66%)	0.31ms (2.74%)	-31.91%
tab5c	no	real	22.88ms	0.74ms (3.25%)	0.23ms (1.03%)	0.00%
		CPU	22.23ms	1.25ms (5.62%)	0.40ms (1.78%)	0.00%
	yes	real	18.70ms	1.88ms (10.06%)	0.59ms (3.18%)	-18.28%
		CPU	18.38ms	1.83ms (9.96%)	0.58ms (3.15%)	-17.35%
tab5d	no	real	11.20ms	0.56ms (5.01%)	0.18ms (1.58%)	0.00%
		CPU	11.03ms	0.47ms (4.25%)	0.15ms (1.34%)	0.00%
	yes	real	8.02ms	0.91ms (11.32%)	0.29ms (3.58%)	-28.43%
		CPU	7.86ms	0.90ms (11.40%)	0.28ms (3.60%)	-28.77%
tab5e	no	real	35.26ms	2.70ms (7.65%)	0.85ms (2.42%)	0.00%
		CPU	34.22ms	2.07ms (6.04%)	0.65ms (1.91%)	0.00%
	yes	real	30.23ms	2.51ms (8.31%)	0.79ms (2.63%)	-14.26%
		CPU	29.48ms	2.23ms (7.55%)	0.70ms (2.39%)	-13.86%
tab5f	no	real	18.42ms	1.52ms (8.28%)	0.48ms (2.62%)	0.00%
		CPU	18.40ms	1.53ms (8.30%)	0.48ms (2.62%)	0.00%
	yes	real	14.21ms	1.26ms (8.88%)	0.40ms (2.81%)	-22.82%
		CPU	14.01ms	1.24ms (8.88%)	0.39ms (2.81%)	-23.89%
tab6a	no	real	195.18ms	6.07ms (3.11%)	1.92ms (0.98%)	0.00%
		CPU	192.32ms	1.88ms (0.98%)	0.59ms (0.31%)	0.00%
	yes	real	188.46ms	2.33ms (1.24%)	0.74ms (0.39%)	-3.44%
		CPU	187.72ms	2.30ms (1.23%)	0.73ms (0.39%)	-2.39%
tab6b	no	real	32.69ms	1.70ms (5.21%)	0.54ms (1.65%)	0.00%
		CPU	32.34ms	1.65ms (5.09%)	0.52ms (1.61%)	0.00%
	yes	real	28.20ms	2.56ms (9.09%)	0.81ms (2.87%)	-13.74%
		CPU	27.29ms	1.83ms (6.69%)	0.58ms (2.12%)	-15.62%
tab6c	no	real	58.83ms	4.40ms (7.48%)	1.39ms (2.36%)	0.00%
		CPU	58.71ms	4.38ms (7.46%)	1.39ms (2.36%)	0.00%
	yes	real	51.30ms	1.94ms (3.78%)	0.61ms (1.20%)	-12.80%
		CPU	50.79ms	2.04ms (4.02%)	0.65ms (1.27%)	-13.48%
tab6d	no	real	33.46ms	2.31ms (6.90%)	0.73ms (2.18%)	0.00%
		CPU	33.20ms	2.18ms (6.58%)	0.69ms (2.08%)	0.00%
	yes	real	31.06ms	4.20ms (13.51%)	1.33ms (4.27%)	-7.17%
		CPU	30.31ms	4.30ms (14.20%)	1.36ms (4.49%)	-8.68%
tab6e	no	real	97.21ms	6.29ms (6.47%)	1.99ms (2.04%)	0.00%
		CPU	96.36ms	4.92ms (5.11%)	1.56ms (1.62%)	0.00%
	yes	real	87.90ms	1.88ms (2.14%)	0.59ms (0.68%)	-9.58%
		CPU	86.31ms	2.95ms (3.42%)	0.93ms (1.08%)	-10.42%
tab6f	no	real	34.28ms	2.56ms (7.47%)	0.81ms (2.36%)	0.00%
		CPU	33.51ms	2.13ms (6.36%)	0.67ms (2.01%)	0.00%
	yes	real	32.34ms	1.84ms (5.68%)	0.58ms (1.80%)	-5.65%
		CPU	31.36ms	1.88ms (5.99%)	0.59ms (1.89%)	-6.42%

tab7a	no	real	309.97ms	4.60ms (1.48%)	1.45ms (0.47%)	0.00%
		CPU	307.75ms	3.91ms (1.27%)	1.24ms (0.40%)	0.00%
	yes	real	324.18ms	14.65ms (4.52%)	4.63ms (1.43%)	4.58%
		CPU	323.04ms	14.59ms (4.52%)	4.61ms (1.43%)	4.97%
tab7b	no	real	82.62ms	4.54ms (5.50%)	1.44ms (1.74%)	0.00%
		CPU	80.67ms	3.72ms (4.61%)	1.18ms (1.46%)	0.00%
	yes	real	81.98ms	8.55ms (10.42%)	2.70ms (3.30%)	-0.78%
		CPU	80.71ms	6.99ms (8.66%)	2.21ms (2.74%)	0.04%
tab7c	no	real	127.31ms	5.69ms (4.47%)	1.80ms (1.41%)	0.00%
		CPU	125.41ms	4.70ms (3.75%)	1.49ms (1.19%)	0.00%
	yes	real	125.31ms	17.40ms (13.89%)	5.50ms (4.39%)	-1.58%
		CPU	123.55ms	17.06ms (13.81%)	5.40ms (4.37%)	-1.49%
tab7d	no	real	70.50ms	0.91ms (1.30%)	0.29ms (0.41%)	0.00%
		CPU	70.37ms	0.92ms (1.31%)	0.29ms (0.42%)	0.00%
	yes	real	65.89ms	0.54ms (0.82%)	0.17ms (0.26%)	-6.54%
		CPU	65.55ms	0.56ms (0.85%)	0.18ms (0.27%)	-6.85%
tab7e	no	real	178.91ms	11.68ms (6.53%)	3.69ms (2.06%)	0.00%
		CPU	176.95ms	9.93ms (5.61%)	3.14ms (1.78%)	0.00%
	yes	real	201.53ms	6.38ms (3.17%)	2.02ms (1.00%)	12.65%
		CPU	200.53ms	6.25ms (3.11%)	1.98ms (0.98%)	13.32%
tab7f	no	real	86.81ms	5.73ms (6.60%)	1.81ms (2.09%)	0.00%
		CPU	83.92ms	3.80ms (4.53%)	1.20ms (1.43%)	0.00%
	yes	real	80.86ms	4.11ms (5.08%)	1.30ms (1.61%)	-6.86%
		CPU	80.34ms	4.18ms (5.20%)	1.32ms (1.65%)	-4.26%
tab8a	no	real	278.40ms	14.96ms (5.37%)	4.73ms (1.70%)	0.00%
		CPU	276.35ms	12.58ms (4.55%)	3.98ms (1.44%)	0.00%
	yes	real	310.06ms	15.88ms (5.12%)	5.02ms (1.62%)	11.37%
		CPU	308.60ms	15.78ms (5.11%)	4.99ms (1.62%)	11.67%
tab8b	no	real	82.01ms	6.10ms (7.44%)	1.93ms (2.35%)	0.00%
		CPU	79.57ms	5.88ms (7.39%)	1.86ms (2.34%)	0.00%
	yes	real	65.42ms	0.91ms (1.39%)	0.29ms (0.44%)	-20.23%
		CPU	65.10ms	0.91ms (1.40%)	0.29ms (0.44%)	-18.18%
tab8c	no	real	109.99ms	3.35ms (3.05%)	1.06ms (0.96%)	0.00%
		CPU	109.78ms	3.31ms (3.02%)	1.05ms (0.95%)	0.00%
	yes	real	115.78ms	17.52ms (15.13%)	5.54ms (4.79%)	5.26%
		CPU	115.35ms	17.44ms (15.12%)	5.51ms (4.78%)	5.07%
tab8d	no	real	83.83ms	3.81ms (4.55%)	1.21ms (1.44%)	0.00%
		CPU	80.59ms	4.53ms (5.62%)	1.43ms (1.78%)	0.00%
	yes	real	71.69ms	5.03ms (7.01%)	1.59ms (2.22%)	-14.48%
		CPU	70.22ms	4.41ms (6.28%)	1.40ms (1.99%)	-12.87%
tab8e	no	real	97.93ms	4.73ms (4.83%)	1.49ms (1.53%)	0.00%
		CPU	97.73ms	4.73ms (4.84%)	1.50ms (1.53%)	0.00%
	yes	real	91.30ms	3.51ms (3.85%)	1.11ms (1.22%)	-6.77%
		CPU	90.93ms	3.51ms (3.86%)	1.11ms (1.22%)	-6.97%
tab8f	no	real	98.01ms	7.73ms (7.89%)	2.44ms (2.49%)	0.00%
		CPU	96.56ms	4.94ms (5.12%)	1.56ms (1.62%)	0.00%
	yes	real	106.16ms	7.82ms (7.37%)	2.47ms (2.33%)	8.32%
		CPU	105.38ms	7.30ms (6.93%)	2.31ms (2.19%)	9.14%

Table A.6: Run times of the model solution program of the problem Tablice kierunkowe (tab) from the finale of XXII Polish Olimpiad in Informatics. For each configuration (Test case and Sandbox columns) the data was collected from 10 runs. Real and CPU times were collected from the same runs. Slowdown is measured from the times of configuration without the sandbox.

A.2.3. Modernizacja autostrady (mod)

Test case	Sandbox	Time	Mean	Std. dev.	Std. err. on the mean	Slowdown
mod0	no	real	5.23ms	0.37ms (7.14%)	0.12ms (2.26%)	0.00%
		CPU	5.20ms	0.28ms (5.44%)	0.09ms (1.72%)	0.00%
	yes	real	5.22ms	0.53ms (10.20%)	0.17ms (3.23%)	-0.33%
		CPU	4.94ms	0.44ms (8.92%)	0.14ms (2.82%)	-5.06%
mod1ocen	no	real	5.16ms	0.32ms (6.14%)	0.10ms (1.94%)	0.00%
		CPU	5.07ms	0.20ms (3.93%)	0.06ms (1.24%)	0.00%
	yes	real	5.14ms	0.31ms (6.01%)	0.10ms (1.90%)	-0.42%
		CPU	4.97ms	0.29ms (5.75%)	0.09ms (1.82%)	-2.02%
mod2ocen	no	real	5.65ms	0.62ms (10.90%)	0.19ms (3.45%)	0.00%
		CPU	5.42ms	0.22ms (4.05%)	0.07ms (1.28%)	0.00%
	yes	real	5.49ms	0.34ms (6.15%)	0.11ms (1.95%)	-2.73%
		CPU	5.30ms	0.35ms (6.65%)	0.11ms (2.10%)	-2.26%
mod3ocen	no	real	84.92ms	2.76ms (3.25%)	0.87ms (1.03%)	0.00%
		CPU	83.99ms	2.60ms (3.09%)	0.82ms (0.98%)	0.00%
	yes	real	80.79ms	0.59ms (0.73%)	0.19ms (0.23%)	-4.87%
		CPU	80.39ms	0.60ms (0.74%)	0.19ms (0.24%)	-4.28%
mod1a	no	real	5.00ms	0.55ms (10.97%)	0.17ms (3.47%)	0.00%
		CPU	4.84ms	0.06ms (1.19%)	0.02ms (0.38%)	0.00%
	yes	real	4.80ms	0.18ms (3.83%)	0.06ms (1.21%)	-4.02%
		CPU	4.63ms	0.18ms (3.90%)	0.06ms (1.23%)	-4.33%
mod1b	no	real	4.84ms	0.06ms (1.19%)	0.02ms (0.38%)	0.00%
		CPU	4.87ms	0.06ms (1.21%)	0.02ms (0.38%)	0.00%
	yes	real	5.01ms	0.29ms (5.71%)	0.09ms (1.81%)	3.52%
		CPU	4.82ms	0.27ms (5.54%)	0.08ms (1.75%)	-1.02%
mod1c	no	real	4.82ms	0.10ms (2.14%)	0.03ms (0.68%)	0.00%
		CPU	4.83ms	0.06ms (1.14%)	0.02ms (0.36%)	0.00%
	yes	real	5.29ms	0.30ms (5.63%)	0.09ms (1.78%)	9.70%
		CPU	5.02ms	0.22ms (4.47%)	0.07ms (1.41%)	3.82%
mod1d	no	real	5.19ms	0.27ms (5.20%)	0.09ms (1.64%)	0.00%
		CPU	5.11ms	0.24ms (4.75%)	0.08ms (1.50%)	0.00%
	yes	real	5.40ms	0.38ms (7.08%)	0.12ms (2.24%)	3.99%
		CPU	5.06ms	0.27ms (5.26%)	0.08ms (1.66%)	-0.92%
mod1e	no	real	5.27ms	0.52ms (9.86%)	0.16ms (3.12%)	0.00%
		CPU	5.09ms	0.23ms (4.56%)	0.07ms (1.44%)	0.00%
	yes	real	5.24ms	0.38ms (7.31%)	0.12ms (2.31%)	-0.57%
		CPU	4.97ms	0.33ms (6.63%)	0.10ms (2.10%)	-2.33%
mod1f	no	real	5.01ms	0.27ms (5.48%)	0.09ms (1.73%)	0.00%
		CPU	5.00ms	0.23ms (4.64%)	0.07ms (1.47%)	0.00%
	yes	real	5.02ms	0.42ms (8.32%)	0.13ms (2.63%)	0.23%
		CPU	4.83ms	0.42ms (8.73%)	0.13ms (2.76%)	-3.30%
mod2a	no	real	5.08ms	0.22ms (4.37%)	0.07ms (1.38%)	0.00%
		CPU	5.05ms	0.21ms (4.17%)	0.07ms (1.32%)	0.00%
	yes	real	5.09ms	0.38ms (7.55%)	0.12ms (2.39%)	0.21%
		CPU	4.92ms	0.36ms (7.38%)	0.11ms (2.33%)	-2.54%
mod2b	no	real	5.34ms	0.51ms (9.64%)	0.16ms (3.05%)	0.00%
		CPU	5.18ms	0.36ms (6.90%)	0.11ms (2.18%)	0.00%
	yes	real	5.03ms	0.33ms (6.59%)	0.10ms (2.09%)	-5.83%
		CPU	4.86ms	0.32ms (6.59%)	0.10ms (2.08%)	-6.08%

mod2c	no	real	5.13ms	0.23ms (4.42%)	0.07ms (1.40%)	0.00%
		CPU	5.13ms	0.20ms (3.86%)	0.06ms (1.22%)	0.00%
	yes	real	5.13ms	0.37ms (7.26%)	0.12ms (2.30%)	-0.04%
		CPU	4.92ms	0.36ms (7.30%)	0.11ms (2.31%)	-4.00%
mod2d	no	real	5.15ms	0.26ms (4.98%)	0.08ms (1.57%)	0.00%
		CPU	5.11ms	0.23ms (4.43%)	0.07ms (1.40%)	0.00%
	yes	real	5.26ms	0.45ms (8.60%)	0.14ms (2.72%)	2.12%
		CPU	4.94ms	0.28ms (5.72%)	0.09ms (1.81%)	-3.28%
mod2e	no	real	5.15ms	0.19ms (3.66%)	0.06ms (1.16%)	0.00%
		CPU	5.10ms	0.19ms (3.67%)	0.06ms (1.16%)	0.00%
	yes	real	5.47ms	0.67ms (12.33%)	0.21ms (3.90%)	6.31%
		CPU	5.05ms	0.31ms (6.14%)	0.10ms (1.94%)	-0.93%
mod2f	no	real	5.31ms	0.21ms (3.98%)	0.07ms (1.26%)	0.00%
		CPU	5.15ms	0.24ms (4.67%)	0.08ms (1.48%)	0.00%
	yes	real	5.18ms	0.41ms (7.87%)	0.13ms (2.49%)	-2.43%
		CPU	4.85ms	0.28ms (5.71%)	0.09ms (1.81%)	-5.83%
mod3a	no	real	5.62ms	0.38ms (6.84%)	0.12ms (2.16%)	0.00%
		CPU	5.41ms	0.24ms (4.51%)	0.08ms (1.42%)	0.00%
	yes	real	5.52ms	0.25ms (4.53%)	0.08ms (1.43%)	-1.70%
		CPU	5.30ms	0.25ms (4.70%)	0.08ms (1.48%)	-2.02%
mod3b	no	real	5.47ms	0.29ms (5.23%)	0.09ms (1.65%)	0.00%
		CPU	5.40ms	0.29ms (5.44%)	0.09ms (1.72%)	0.00%
	yes	real	5.65ms	0.43ms (7.68%)	0.14ms (2.43%)	3.26%
		CPU	5.25ms	0.32ms (6.03%)	0.10ms (1.91%)	-2.81%
mod3c	no	real	5.73ms	0.47ms (8.27%)	0.15ms (2.62%)	0.00%
		CPU	5.43ms	0.24ms (4.46%)	0.08ms (1.41%)	0.00%
	yes	real	5.50ms	0.58ms (10.58%)	0.18ms (3.35%)	-3.99%
		CPU	5.10ms	0.33ms (6.47%)	0.10ms (2.04%)	-6.04%
mod3d	no	real	5.67ms	0.46ms (8.09%)	0.14ms (2.56%)	0.00%
		CPU	5.58ms	0.29ms (5.11%)	0.09ms (1.62%)	0.00%
	yes	real	5.40ms	0.30ms (5.55%)	0.09ms (1.76%)	-4.76%
		CPU	5.18ms	0.29ms (5.54%)	0.09ms (1.75%)	-7.14%
mod3e	no	real	5.54ms	0.29ms (5.24%)	0.09ms (1.66%)	0.00%
		CPU	5.54ms	0.30ms (5.36%)	0.09ms (1.69%)	0.00%
	yes	real	5.63ms	0.53ms (9.47%)	0.17ms (2.99%)	1.61%
		CPU	5.31ms	0.36ms (6.77%)	0.11ms (2.14%)	-4.14%
mod3f	no	real	5.67ms	0.38ms (6.71%)	0.12ms (2.12%)	0.00%
		CPU	5.64ms	0.34ms (6.05%)	0.11ms (1.91%)	0.00%
	yes	real	5.63ms	0.64ms (11.31%)	0.20ms (3.58%)	-0.61%
		CPU	5.20ms	0.37ms (7.08%)	0.12ms (2.24%)	-7.83%
mod3g	no	real	5.69ms	0.33ms (5.73%)	0.10ms (1.81%)	0.00%
		CPU	5.57ms	0.21ms (3.80%)	0.07ms (1.20%)	0.00%
	yes	real	5.51ms	0.35ms (6.28%)	0.11ms (1.99%)	-3.06%
		CPU	5.31ms	0.35ms (6.63%)	0.11ms (2.10%)	-4.52%
mod4a	no	real	18.07ms	1.03ms (5.70%)	0.33ms (1.80%)	0.00%
		CPU	17.61ms	0.53ms (2.99%)	0.17ms (0.94%)	0.00%
	yes	real	17.97ms	0.74ms (4.10%)	0.23ms (1.30%)	-0.55%
		CPU	17.54ms	0.61ms (3.47%)	0.19ms (1.10%)	-0.42%
mod4b	no	real	14.61ms	1.01ms (6.89%)	0.32ms (2.18%)	0.00%
		CPU	14.09ms	0.80ms (5.70%)	0.25ms (1.80%)	0.00%
	yes	real	13.69ms	0.82ms (6.00%)	0.26ms (1.90%)	-6.26%
		CPU	13.07ms	0.42ms (3.21%)	0.13ms (1.01%)	-7.23%

mod4c	no	real	19.38ms	1.10ms (5.70%)	0.35ms (1.80%)	0.00%
		CPU	18.96ms	0.76ms (4.02%)	0.24ms (1.27%)	0.00%
	yes	real	19.63ms	1.21ms (6.18%)	0.38ms (1.95%)	1.28%
		CPU	18.99ms	0.94ms (4.94%)	0.30ms (1.56%)	0.13%
mod4d	no	real	18.79ms	1.31ms (6.95%)	0.41ms (2.20%)	0.00%
		CPU	18.05ms	0.61ms (3.39%)	0.19ms (1.07%)	0.00%
	yes	real	18.49ms	0.96ms (5.22%)	0.30ms (1.65%)	-1.61%
		CPU	18.22ms	0.94ms (5.15%)	0.30ms (1.63%)	0.98%
mod4e	no	real	17.26ms	0.75ms (4.33%)	0.24ms (1.37%)	0.00%
		CPU	16.66ms	0.65ms (3.92%)	0.21ms (1.24%)	0.00%
	yes	real	16.96ms	0.70ms (4.13%)	0.22ms (1.31%)	-1.75%
		CPU	16.61ms	0.58ms (3.49%)	0.18ms (1.10%)	-0.26%
mod4f	no	real	17.29ms	0.56ms (3.24%)	0.18ms (1.02%)	0.00%
		CPU	16.88ms	0.45ms (2.67%)	0.14ms (0.85%)	0.00%
	yes	real	16.97ms	0.94ms (5.56%)	0.30ms (1.76%)	-1.84%
		CPU	16.34ms	0.60ms (3.65%)	0.19ms (1.15%)	-3.17%
mod5a	no	real	47.76ms	3.49ms (7.30%)	1.10ms (2.31%)	0.00%
		CPU	46.49ms	2.25ms (4.84%)	0.71ms (1.53%)	0.00%
	yes	real	47.47ms	2.41ms (5.07%)	0.76ms (1.60%)	-0.60%
		CPU	45.91ms	1.61ms (3.51%)	0.51ms (1.11%)	-1.25%
mod5b	no	real	36.91ms	1.45ms (3.92%)	0.46ms (1.24%)	0.00%
		CPU	36.60ms	1.70ms (4.66%)	0.54ms (1.47%)	0.00%
	yes	real	40.61ms	4.71ms (11.59%)	1.49ms (3.67%)	10.01%
		CPU	38.87ms	2.61ms (6.71%)	0.82ms (2.12%)	6.21%
mod5c	no	real	46.00ms	2.01ms (4.38%)	0.64ms (1.38%)	0.00%
		CPU	45.47ms	2.04ms (4.50%)	0.65ms (1.42%)	0.00%
	yes	real	48.42ms	2.99ms (6.17%)	0.95ms (1.95%)	5.26%
		CPU	47.85ms	2.88ms (6.02%)	0.91ms (1.90%)	5.23%
mod5d	no	real	48.50ms	1.95ms (4.02%)	0.62ms (1.27%)	0.00%
		CPU	47.81ms	1.67ms (3.50%)	0.53ms (1.11%)	0.00%
	yes	real	49.03ms	2.97ms (6.05%)	0.94ms (1.91%)	1.10%
		CPU	46.35ms	1.69ms (3.65%)	0.53ms (1.15%)	-3.07%
mod5e	no	real	42.90ms	0.52ms (1.21%)	0.16ms (0.38%)	0.00%
		CPU	42.57ms	0.50ms (1.17%)	0.16ms (0.37%)	0.00%
	yes	real	42.55ms	0.98ms (2.30%)	0.31ms (0.73%)	-0.82%
		CPU	42.25ms	0.99ms (2.35%)	0.31ms (0.74%)	-0.75%
mod5f	no	real	41.63ms	0.69ms (1.65%)	0.22ms (0.52%)	0.00%
		CPU	41.33ms	0.29ms (0.69%)	0.09ms (0.22%)	0.00%
	yes	real	42.25ms	1.29ms (3.05%)	0.41ms (0.96%)	1.49%
		CPU	41.71ms	0.69ms (1.66%)	0.22ms (0.53%)	0.90%
mod5g	no	real	47.79ms	2.75ms (5.75%)	0.87ms (1.82%)	0.00%
		CPU	46.26ms	1.46ms (3.15%)	0.46ms (1.00%)	0.00%
	yes	real	48.65ms	2.94ms (6.04%)	0.93ms (1.91%)	1.81%
		CPU	45.95ms	1.92ms (4.18%)	0.61ms (1.32%)	-0.66%
mod6a	no	real	84.97ms	3.24ms (3.81%)	1.03ms (1.21%)	0.00%
		CPU	83.63ms	2.82ms (3.38%)	0.89ms (1.07%)	0.00%
	yes	real	77.73ms	2.91ms (3.74%)	0.92ms (1.18%)	-8.52%
		CPU	76.94ms	2.52ms (3.28%)	0.80ms (1.04%)	-8.00%
mod6b	no	real	58.88ms	0.62ms (1.05%)	0.20ms (0.33%)	0.00%
		CPU	58.47ms	0.34ms (0.58%)	0.11ms (0.18%)	0.00%
	yes	real	64.92ms	5.16ms (7.95%)	1.63ms (2.51%)	10.26%
		CPU	61.48ms	2.47ms (4.02%)	0.78ms (1.27%)	5.16%

mod6c	no	real	82.99ms	3.87ms (4.67%)	1.23ms (1.48%)	0.00%
		CPU	81.27ms	3.42ms (4.21%)	1.08ms (1.33%)	0.00%
	yes	real	81.73ms	4.41ms (5.40%)	1.40ms (1.71%)	-1.52%
		CPU	80.23ms	2.32ms (2.89%)	0.73ms (0.91%)	-1.29%
mod6d	no	real	91.40ms	3.39ms (3.71%)	1.07ms (1.17%)	0.00%
		CPU	89.29ms	2.80ms (3.14%)	0.89ms (0.99%)	0.00%
	yes	real	92.28ms	3.39ms (3.68%)	1.07ms (1.16%)	0.96%
		CPU	88.44ms	2.28ms (2.57%)	0.72ms (0.81%)	-0.96%
mod6e	no	real	95.71ms	3.64ms (3.80%)	1.15ms (1.20%)	0.00%
		CPU	94.29ms	3.19ms (3.38%)	1.01ms (1.07%)	0.00%
	yes	real	88.47ms	3.05ms (3.45%)	0.96ms (1.09%)	-7.57%
		CPU	87.98ms	3.03ms (3.45%)	0.96ms (1.09%)	-6.69%
mod6f	no	real	72.95ms	0.90ms (1.23%)	0.28ms (0.39%)	0.00%
		CPU	72.55ms	0.58ms (0.80%)	0.18ms (0.25%)	0.00%
	yes	real	77.42ms	4.42ms (5.71%)	1.40ms (1.81%)	6.13%
		CPU	75.77ms	3.84ms (5.07%)	1.21ms (1.60%)	4.45%
mod6g	no	real	77.06ms	5.44ms (7.06%)	1.72ms (2.23%)	0.00%
		CPU	75.39ms	3.80ms (5.03%)	1.20ms (1.59%)	0.00%
	yes	real	70.94ms	0.71ms (1.00%)	0.22ms (0.32%)	-7.95%
		CPU	70.49ms	0.71ms (1.01%)	0.22ms (0.32%)	-6.50%
mod7a	no	real	143.37ms	5.21ms (3.63%)	1.65ms (1.15%)	0.00%
		CPU	142.65ms	5.22ms (3.66%)	1.65ms (1.16%)	0.00%
	yes	real	154.70ms	3.71ms (2.40%)	1.17ms (0.76%)	7.91%
		CPU	152.51ms	1.71ms (1.12%)	0.54ms (0.36%)	6.91%
mod7b	no	real	109.98ms	3.10ms (2.82%)	0.98ms (0.89%)	0.00%
		CPU	108.58ms	2.19ms (2.02%)	0.69ms (0.64%)	0.00%
	yes	real	109.62ms	2.86ms (2.61%)	0.90ms (0.82%)	-0.33%
		CPU	108.69ms	2.80ms (2.57%)	0.88ms (0.81%)	0.10%
mod7c	no	real	113.99ms	2.30ms (2.02%)	0.73ms (0.64%)	0.00%
		CPU	113.21ms	2.00ms (1.76%)	0.63ms (0.56%)	0.00%
	yes	real	115.04ms	1.55ms (1.35%)	0.49ms (0.43%)	0.93%
		CPU	114.31ms	1.54ms (1.35%)	0.49ms (0.43%)	0.97%
mod7d	no	real	147.84ms	7.48ms (5.06%)	2.36ms (1.60%)	0.00%
		CPU	145.09ms	5.84ms (4.03%)	1.85ms (1.27%)	0.00%
	yes	real	137.64ms	0.90ms (0.66%)	0.29ms (0.21%)	-6.90%
		CPU	136.81ms	0.94ms (0.69%)	0.30ms (0.22%)	-5.71%
mod7e	no	real	156.91ms	3.85ms (2.45%)	1.22ms (0.78%)	0.00%
		CPU	155.96ms	4.45ms (2.85%)	1.41ms (0.90%)	0.00%
	yes	real	160.70ms	3.76ms (2.34%)	1.19ms (0.74%)	2.41%
		CPU	158.09ms	2.22ms (1.40%)	0.70ms (0.44%)	1.37%
mod7f	no	real	159.59ms	3.35ms (2.10%)	1.06ms (0.66%)	0.00%
		CPU	156.29ms	2.94ms (1.88%)	0.93ms (0.60%)	0.00%
	yes	real	148.48ms	6.94ms (4.67%)	2.19ms (1.48%)	-6.96%
		CPU	146.50ms	3.49ms (2.38%)	1.10ms (0.75%)	-6.27%
mod8a	no	real	304.65ms	5.46ms (1.79%)	1.73ms (0.57%)	0.00%
		CPU	301.59ms	2.08ms (0.69%)	0.66ms (0.22%)	0.00%
	yes	real	289.97ms	13.63ms (4.70%)	4.31ms (1.49%)	-4.82%
		CPU	288.48ms	13.57ms (4.70%)	4.29ms (1.49%)	-4.35%
mod8b	no	real	236.16ms	7.83ms (3.32%)	2.48ms (1.05%)	0.00%
		CPU	233.15ms	6.83ms (2.93%)	2.16ms (0.93%)	0.00%
	yes	real	230.26ms	11.56ms (5.02%)	3.66ms (1.59%)	-2.50%
		CPU	228.55ms	11.53ms (5.05%)	3.65ms (1.60%)	-1.97%

mod8c	no	real	225.84ms	5.98ms (2.65%)	1.89ms (0.84%)	0.00%
		CPU	224.34ms	5.50ms (2.45%)	1.74ms (0.78%)	0.00%
	yes	real	242.43ms	10.08ms (4.16%)	3.19ms (1.32%)	7.35%
		CPU	241.09ms	10.05ms (4.17%)	3.18ms (1.32%)	7.47%
mod8d	no	real	314.94ms	2.72ms (0.86%)	0.86ms (0.27%)	0.00%
		CPU	313.13ms	2.33ms (0.74%)	0.74ms (0.24%)	0.00%
	yes	real	317.00ms	6.24ms (1.97%)	1.97ms (0.62%)	0.65%
		CPU	315.12ms	6.34ms (2.01%)	2.00ms (0.64%)	0.64%
mod8e	no	real	358.78ms	19.81ms (5.52%)	6.26ms (1.75%)	0.00%
		CPU	357.30ms	19.82ms (5.55%)	6.27ms (1.75%)	0.00%
	yes	real	353.70ms	38.76ms (10.96%)	12.26ms (3.47%)	-1.42%
		CPU	350.86ms	37.16ms (10.59%)	11.75ms (3.35%)	-1.80%
mod8f	no	real	326.37ms	7.80ms (2.39%)	2.47ms (0.76%)	0.00%
		CPU	322.44ms	4.22ms (1.31%)	1.33ms (0.41%)	0.00%
	yes	real	327.56ms	5.16ms (1.58%)	1.63ms (0.50%)	0.36%
		CPU	325.80ms	5.12ms (1.57%)	1.62ms (0.50%)	1.04%
mod8g	no	real	325.61ms	8.25ms (2.53%)	2.61ms (0.80%)	0.00%
		CPU	318.04ms	4.06ms (1.28%)	1.28ms (0.40%)	0.00%
	yes	real	330.21ms	11.28ms (3.42%)	3.57ms (1.08%)	1.41%
		CPU	328.40ms	11.27ms (3.43%)	3.56ms (1.09%)	3.26%
mod9a	no	real	490.02ms	8.21ms (1.68%)	2.60ms (0.53%)	0.00%
		CPU	482.27ms	6.26ms (1.30%)	1.98ms (0.41%)	0.00%
	yes	real	485.52ms	5.65ms (1.16%)	1.79ms (0.37%)	-0.92%
		CPU	482.72ms	5.62ms (1.16%)	1.78ms (0.37%)	0.09%
mod9b	no	real	349.57ms	28.07ms (8.03%)	8.88ms (2.54%)	0.00%
		CPU	345.96ms	29.18ms (8.43%)	9.23ms (2.67%)	0.00%
	yes	real	344.45ms	38.38ms (11.14%)	12.14ms (3.52%)	-1.46%
		CPU	342.58ms	38.09ms (11.12%)	12.05ms (3.52%)	-0.98%
mod9c	no	real	435.37ms	8.38ms (1.92%)	2.65ms (0.61%)	0.00%
		CPU	431.75ms	5.03ms (1.16%)	1.59ms (0.37%)	0.00%
	yes	real	406.78ms	49.95ms (12.28%)	15.80ms (3.88%)	-6.57%
		CPU	404.56ms	49.75ms (12.30%)	15.73ms (3.89%)	-6.30%
mod9d	no	real	450.48ms	49.69ms (11.03%)	15.71ms (3.49%)	0.00%
		CPU	446.09ms	45.83ms (10.27%)	14.49ms (3.25%)	0.00%
	yes	real	516.08ms	10.35ms (2.01%)	3.27ms (0.63%)	14.56%
		CPU	508.79ms	13.24ms (2.60%)	4.19ms (0.82%)	14.05%
mod9e	no	real	509.04ms	8.47ms (1.66%)	2.68ms (0.53%)	0.00%
		CPU	505.00ms	4.30ms (0.85%)	1.36ms (0.27%)	0.00%
	yes	real	479.69ms	41.40ms (8.63%)	13.09ms (2.73%)	-5.77%
		CPU	475.50ms	41.49ms (8.73%)	13.12ms (2.76%)	-5.84%
mod9f	no	real	382.76ms	36.53ms (9.54%)	11.55ms (3.02%)	0.00%
		CPU	379.94ms	33.86ms (8.91%)	10.71ms (2.82%)	0.00%
	yes	real	442.34ms	28.82ms (6.51%)	9.11ms (2.06%)	15.57%
		CPU	437.64ms	26.43ms (6.04%)	8.36ms (1.91%)	15.19%
mod10a	no	real	546.64ms	41.89ms (7.66%)	13.25ms (2.42%)	0.00%
		CPU	541.98ms	40.12ms (7.40%)	12.69ms (2.34%)	0.00%
	yes	real	553.07ms	46.09ms (8.33%)	14.57ms (2.64%)	1.17%
		CPU	545.55ms	43.08ms (7.90%)	13.62ms (2.50%)	0.66%
mod10b	no	real	420.02ms	9.52ms (2.27%)	3.01ms (0.72%)	0.00%
		CPU	413.88ms	7.03ms (1.70%)	2.22ms (0.54%)	0.00%
	yes	real	419.49ms	11.06ms (2.64%)	3.50ms (0.83%)	-0.12%
		CPU	416.90ms	10.99ms (2.64%)	3.48ms (0.83%)	0.73%

mod10c	no	real	488.18ms	10.60ms (2.17%)	3.35ms (0.69%)	0.00%
		CPU	482.89ms	10.71ms (2.22%)	3.39ms (0.70%)	0.00%
	yes	real	482.24ms	4.57ms (0.95%)	1.44ms (0.30%)	-1.22%
		CPU	478.63ms	4.50ms (0.94%)	1.42ms (0.30%)	-0.88%
mod10d	no	real	536.97ms	5.33ms (0.99%)	1.69ms (0.31%)	0.00%
		CPU	533.12ms	4.19ms (0.79%)	1.33ms (0.25%)	0.00%
	yes	real	520.43ms	38.75ms (7.45%)	12.25ms (2.35%)	-3.08%
		CPU	517.93ms	38.47ms (7.43%)	12.17ms (2.35%)	-2.85%
mod10e	no	real	526.14ms	43.07ms (8.19%)	13.62ms (2.59%)	0.00%
		CPU	519.83ms	40.43ms (7.78%)	12.78ms (2.46%)	0.00%
	yes	real	544.12ms	6.10ms (1.12%)	1.93ms (0.35%)	3.42%
		CPU	541.52ms	6.05ms (1.12%)	1.91ms (0.35%)	4.17%
mod10f	no	real	554.25ms	8.12ms (1.47%)	2.57ms (0.46%)	0.00%
		CPU	546.83ms	4.93ms (0.90%)	1.56ms (0.29%)	0.00%
	yes	real	549.90ms	2.12ms (0.39%)	0.67ms (0.12%)	-0.79%
		CPU	547.10ms	2.06ms (0.38%)	0.65ms (0.12%)	0.05%
mod10g	no	real	523.16ms	7.04ms (1.35%)	2.23ms (0.43%)	0.00%
		CPU	520.59ms	7.07ms (1.36%)	2.24ms (0.43%)	0.00%
	yes	real	515.51ms	6.05ms (1.17%)	1.91ms (0.37%)	-1.46%
		CPU	512.87ms	6.00ms (1.17%)	1.90ms (0.37%)	-1.48%

Table A.7: Run times of the model solution program of the problem Modernizacja autostrady (mod) from the finale of XXII Polish Olimpiad in Informatics. For each configuration (Test case and Sandbox columns) the data was collected from 10 runs. Real and CPU times were collected from the same runs. Slowdown is measured from the times of configuration without the sandbox.

A.2.4. Wycieczki (wyc)

Test case	Sandbox	Time	Mean	Std. dev.	Std. err. on the mean	Slowdown
wyc0	no	real	1.28ms	0.03ms (2.27%)	0.01ms (0.72%)	0.00%
		CPU	1.35ms	0.04ms (2.73%)	0.01ms (0.86%)	0.00%
	yes	real	0.99ms	0.31ms (31.77%)	0.10ms (10.05%)	-23.23%
		CPU	0.89ms	0.27ms (30.10%)	0.08ms (9.52%)	-34.24%
wyc1ocen	no	real	1.96ms	0.59ms (30.03%)	0.19ms (9.50%)	0.00%
		CPU	1.95ms	0.60ms (30.99%)	0.19ms (9.80%)	0.00%
	yes	real	1.48ms	0.64ms (42.91%)	0.20ms (13.57%)	-24.48%
		CPU	1.38ms	0.61ms (44.11%)	0.19ms (13.95%)	-29.03%
wyc2ocen	no	real	7.88ms	0.57ms (7.21%)	0.18ms (2.28%)	0.00%
		CPU	7.97ms	0.67ms (8.44%)	0.21ms (2.67%)	0.00%
	yes	real	7.42ms	1.13ms (15.29%)	0.36ms (4.84%)	-5.95%
		CPU	6.88ms	0.78ms (11.28%)	0.25ms (3.57%)	-13.68%
wyc1a	no	real	1.64ms	0.56ms (33.78%)	0.18ms (10.68%)	0.00%
		CPU	1.71ms	0.55ms (32.21%)	0.17ms (10.19%)	0.00%
	yes	real	1.35ms	0.60ms (44.76%)	0.19ms (14.16%)	-17.86%
		CPU	1.15ms	0.42ms (36.29%)	0.13ms (11.48%)	-32.39%
wyc1b	no	real	1.41ms	0.47ms (33.20%)	0.15ms (10.50%)	0.00%
		CPU	1.46ms	0.45ms (30.44%)	0.14ms (9.63%)	0.00%
	yes	real	1.26ms	0.68ms (54.41%)	0.22ms (17.20%)	-10.61%
		CPU	1.16ms	0.64ms (55.05%)	0.20ms (17.41%)	-20.86%

wyc1c	no	real	1.45ms	0.03ms (2.09%)	0.01ms (0.66%)	0.00%
		CPU	1.51ms	0.03ms (2.27%)	0.01ms (0.72%)	0.00%
	yes	real	1.16ms	0.34ms (29.76%)	0.11ms (9.41%)	-19.89%
		CPU	1.04ms	0.29ms (27.41%)	0.09ms (8.67%)	-31.16%
wyc1d	no	real	1.56ms	0.26ms (16.44%)	0.08ms (5.20%)	0.00%
		CPU	1.69ms	0.34ms (20.26%)	0.11ms (6.41%)	0.00%
	yes	real	1.18ms	0.31ms (25.81%)	0.10ms (8.16%)	-24.15%
		CPU	1.07ms	0.30ms (28.12%)	0.10ms (8.89%)	-36.72%
wyc1e	no	real	1.75ms	0.64ms (36.47%)	0.20ms (11.53%)	0.00%
		CPU	1.70ms	0.30ms (17.57%)	0.09ms (5.56%)	0.00%
	yes	real	1.25ms	0.41ms (32.44%)	0.13ms (10.26%)	-28.34%
		CPU	1.06ms	0.31ms (29.46%)	0.10ms (9.32%)	-37.49%
wyc2a	no	real	2.44ms	0.60ms (24.49%)	0.19ms (7.74%)	0.00%
		CPU	2.49ms	0.58ms (23.27%)	0.18ms (7.36%)	0.00%
	yes	real	2.29ms	0.79ms (34.33%)	0.25ms (10.86%)	-5.99%
		CPU	1.84ms	0.55ms (29.91%)	0.17ms (9.46%)	-26.01%
wyc2b	no	real	3.09ms	0.63ms (20.45%)	0.20ms (6.47%)	0.00%
		CPU	3.22ms	0.62ms (19.20%)	0.20ms (6.07%)	0.00%
	yes	real	2.64ms	0.62ms (23.47%)	0.20ms (7.42%)	-14.55%
		CPU	2.40ms	0.60ms (25.20%)	0.19ms (7.97%)	-25.43%
wyc2c	no	real	3.82ms	0.81ms (21.18%)	0.26ms (6.70%)	0.00%
		CPU	3.83ms	0.70ms (18.19%)	0.22ms (5.75%)	0.00%
	yes	real	3.45ms	0.80ms (23.22%)	0.25ms (7.34%)	-9.58%
		CPU	3.32ms	0.77ms (23.25%)	0.24ms (7.35%)	-13.27%
wyc2d	no	real	2.92ms	0.49ms (16.83%)	0.16ms (5.32%)	0.00%
		CPU	2.92ms	0.36ms (12.49%)	0.12ms (3.95%)	0.00%
	yes	real	2.38ms	0.41ms (17.05%)	0.13ms (5.39%)	-18.52%
		CPU	2.26ms	0.36ms (16.03%)	0.11ms (5.07%)	-22.69%
wyc2e	no	real	3.27ms	0.50ms (15.35%)	0.16ms (4.85%)	0.00%
		CPU	3.28ms	0.50ms (15.26%)	0.16ms (4.83%)	0.00%
	yes	real	2.95ms	0.60ms (20.40%)	0.19ms (6.45%)	-9.94%
		CPU	2.81ms	0.60ms (21.42%)	0.19ms (6.77%)	-14.11%
wyc3a	no	real	4.48ms	0.82ms (18.37%)	0.26ms (5.81%)	0.00%
		CPU	4.53ms	0.79ms (17.47%)	0.25ms (5.52%)	0.00%
	yes	real	4.08ms	0.92ms (22.52%)	0.29ms (7.12%)	-9.07%
		CPU	3.84ms	0.84ms (21.80%)	0.26ms (6.89%)	-15.27%
wyc3b	no	real	2.00ms	0.64ms (32.28%)	0.20ms (10.21%)	0.00%
		CPU	2.06ms	0.63ms (30.54%)	0.20ms (9.66%)	0.00%
	yes	real	1.26ms	0.05ms (3.84%)	0.02ms (1.21%)	-37.08%
		CPU	1.15ms	0.02ms (2.07%)	0.01ms (0.66%)	-44.41%
wyc3c	no	real	9.61ms	0.88ms (9.13%)	0.28ms (2.89%)	0.00%
		CPU	9.50ms	0.89ms (9.34%)	0.28ms (2.95%)	0.00%
	yes	real	8.70ms	1.34ms (15.38%)	0.42ms (4.86%)	-9.41%
		CPU	8.27ms	0.90ms (10.85%)	0.28ms (3.43%)	-12.89%
wyc3d	no	real	7.42ms	0.99ms (13.40%)	0.31ms (4.24%)	0.00%
		CPU	7.33ms	0.83ms (11.37%)	0.26ms (3.60%)	0.00%
	yes	real	6.87ms	0.89ms (12.96%)	0.28ms (4.10%)	-7.47%
		CPU	6.67ms	0.84ms (12.63%)	0.27ms (3.99%)	-9.01%
wyc3e	no	real	6.95ms	0.90ms (12.99%)	0.29ms (4.11%)	0.00%
		CPU	6.97ms	0.90ms (12.92%)	0.28ms (4.09%)	0.00%
	yes	real	6.49ms	0.78ms (12.01%)	0.25ms (3.80%)	-6.68%
		CPU	6.34ms	0.77ms (12.18%)	0.24ms (3.85%)	-8.93%

wyc4a	no	real	3.50ms	0.57ms (16.30%)	0.18ms (5.16%)	0.00%
		CPU	3.55ms	0.56ms (15.74%)	0.18ms (4.98%)	0.00%
	yes	real	3.62ms	0.91ms (25.19%)	0.29ms (7.97%)	3.46%
		CPU	3.47ms	0.88ms (25.46%)	0.28ms (8.05%)	-2.34%
wyc4b	no	real	1.55ms	0.39ms (25.51%)	0.12ms (8.07%)	0.00%
		CPU	1.60ms	0.39ms (24.22%)	0.12ms (7.66%)	0.00%
	yes	real	1.35ms	0.64ms (47.20%)	0.20ms (14.93%)	-12.90%
		CPU	1.21ms	0.63ms (51.86%)	0.20ms (16.40%)	-24.27%
wyc4c	no	real	7.97ms	0.82ms (10.28%)	0.26ms (3.25%)	0.00%
		CPU	7.69ms	0.67ms (8.69%)	0.21ms (2.75%)	0.00%
	yes	real	6.99ms	1.01ms (14.43%)	0.32ms (4.56%)	-12.23%
		CPU	6.74ms	0.98ms (14.53%)	0.31ms (4.60%)	-12.27%
wyc4d	no	real	9.32ms	0.72ms (7.76%)	0.23ms (2.45%)	0.00%
		CPU	9.21ms	0.66ms (7.15%)	0.21ms (2.26%)	0.00%
	yes	real	8.69ms	0.97ms (11.18%)	0.31ms (3.54%)	-6.82%
		CPU	8.42ms	0.82ms (9.78%)	0.26ms (3.09%)	-8.49%
wyc4e	no	real	6.77ms	0.79ms (11.62%)	0.25ms (3.67%)	0.00%
		CPU	6.65ms	0.72ms (10.86%)	0.23ms (3.43%)	0.00%
	yes	real	5.83ms	0.81ms (13.98%)	0.26ms (4.42%)	-13.92%
		CPU	5.49ms	0.63ms (11.48%)	0.20ms (3.63%)	-17.37%
wyc5a	no	real	4.09ms	0.95ms (23.33%)	0.30ms (7.38%)	0.00%
		CPU	3.96ms	0.71ms (17.88%)	0.22ms (5.65%)	0.00%
	yes	real	3.24ms	0.38ms (11.58%)	0.12ms (3.66%)	-20.70%
		CPU	3.10ms	0.38ms (12.21%)	0.12ms (3.86%)	-21.78%
wyc5b	no	real	6.28ms	0.77ms (12.33%)	0.24ms (3.90%)	0.00%
		CPU	6.28ms	0.77ms (12.22%)	0.24ms (3.87%)	0.00%
	yes	real	5.62ms	0.64ms (11.45%)	0.20ms (3.62%)	-10.40%
		CPU	5.47ms	0.64ms (11.62%)	0.20ms (3.68%)	-12.89%
wyc5c	no	real	14.23ms	1.15ms (8.10%)	0.36ms (2.56%)	0.00%
		CPU	13.85ms	1.11ms (8.02%)	0.35ms (2.53%)	0.00%
	yes	real	13.46ms	1.35ms (10.00%)	0.43ms (3.16%)	-5.41%
		CPU	13.24ms	1.31ms (9.88%)	0.41ms (3.12%)	-4.39%
wyc5d	no	real	14.06ms	1.52ms (10.81%)	0.48ms (3.42%)	0.00%
		CPU	13.60ms	1.24ms (9.11%)	0.39ms (2.88%)	0.00%
	yes	real	13.50ms	1.32ms (9.78%)	0.42ms (3.09%)	-3.95%
		CPU	13.24ms	1.22ms (9.19%)	0.38ms (2.91%)	-2.63%
wyc5e	no	real	10.82ms	1.01ms (9.37%)	0.32ms (2.96%)	0.00%
		CPU	10.60ms	0.96ms (9.02%)	0.30ms (2.85%)	0.00%
	yes	real	10.53ms	1.47ms (13.99%)	0.47ms (4.42%)	-2.71%
		CPU	10.32ms	1.40ms (13.60%)	0.44ms (4.30%)	-2.65%
wyc6a	no	real	5.70ms	0.86ms (15.06%)	0.27ms (4.76%)	0.00%
		CPU	5.31ms	0.54ms (10.12%)	0.17ms (3.20%)	0.00%
	yes	real	4.41ms	0.03ms (0.71%)	0.01ms (0.22%)	-22.70%
		CPU	4.27ms	0.01ms (0.29%)	0.00ms (0.09%)	-19.49%
wyc6b	no	real	3.14ms	0.01ms (0.32%)	0.00ms (0.10%)	0.00%
		CPU	3.20ms	0.01ms (0.31%)	0.00ms (0.10%)	0.00%
	yes	real	2.74ms	0.06ms (2.19%)	0.02ms (0.69%)	-12.95%
		CPU	2.63ms	0.06ms (2.22%)	0.02ms (0.70%)	-17.63%
wyc6c	no	real	13.34ms	0.06ms (0.45%)	0.02ms (0.14%)	0.00%
		CPU	13.33ms	0.06ms (0.48%)	0.02ms (0.15%)	0.00%
	yes	real	12.97ms	0.05ms (0.39%)	0.02ms (0.12%)	-2.76%
		CPU	12.76ms	0.02ms (0.15%)	0.01ms (0.05%)	-4.27%

wyc6d	no	real	15.40ms	0.01ms (0.10%)	0.00ms (0.03%)	0.00%
		CPU	15.39ms	0.01ms (0.09%)	0.00ms (0.03%)	0.00%
	yes	real	15.00ms	0.05ms (0.34%)	0.02ms (0.11%)	-2.62%
		CPU	14.82ms	0.02ms (0.15%)	0.01ms (0.05%)	-3.71%
wyc6e	no	real	11.31ms	0.04ms (0.34%)	0.01ms (0.11%)	0.00%
		CPU	11.31ms	0.09ms (0.76%)	0.03ms (0.24%)	0.00%
	yes	real	10.89ms	0.03ms (0.26%)	0.01ms (0.08%)	-3.75%
		CPU	10.73ms	0.01ms (0.08%)	0.00ms (0.03%)	-5.16%
wyc7a	no	real	2.05ms	0.01ms (0.59%)	0.00ms (0.19%)	0.00%
		CPU	2.11ms	0.01ms (0.39%)	0.00ms (0.12%)	0.00%
	yes	real	1.62ms	0.02ms (1.35%)	0.01ms (0.43%)	-20.60%
		CPU	1.53ms	0.01ms (0.88%)	0.00ms (0.28%)	-27.33%
wyc7b	no	real	1.31ms	0.01ms (0.65%)	0.00ms (0.21%)	0.00%
		CPU	1.35ms	0.05ms (3.37%)	0.01ms (1.07%)	0.00%
	yes	real	0.91ms	0.04ms (4.64%)	0.01ms (1.47%)	-31.04%
		CPU	0.82ms	0.04ms (5.12%)	0.01ms (1.62%)	-39.27%
wyc7c	no	real	2.93ms	0.01ms (0.41%)	0.00ms (0.13%)	0.00%
		CPU	2.99ms	0.01ms (0.27%)	0.00ms (0.08%)	0.00%
	yes	real	2.41ms	0.06ms (2.46%)	0.02ms (0.78%)	-17.56%
		CPU	2.30ms	0.02ms (0.71%)	0.01ms (0.22%)	-22.98%
wyc7d	no	real	4.15ms	0.53ms (12.73%)	0.17ms (4.03%)	0.00%
		CPU	4.19ms	0.52ms (12.30%)	0.16ms (3.89%)	0.00%
	yes	real	3.63ms	0.40ms (10.99%)	0.13ms (3.48%)	-12.58%
		CPU	3.51ms	0.39ms (11.22%)	0.12ms (3.55%)	-16.30%
wyc7e	no	real	3.46ms	0.69ms (19.79%)	0.22ms (6.26%)	0.00%
		CPU	3.42ms	0.68ms (19.88%)	0.21ms (6.29%)	0.00%
	yes	real	3.60ms	0.88ms (24.55%)	0.28ms (7.76%)	4.06%
		CPU	3.24ms	0.81ms (24.91%)	0.26ms (7.88%)	-5.18%
wyc8a	no	real	4.37ms	0.85ms (19.37%)	0.27ms (6.13%)	0.00%
		CPU	4.41ms	0.83ms (18.81%)	0.26ms (5.95%)	0.00%
	yes	real	3.53ms	0.64ms (18.25%)	0.20ms (5.77%)	-19.23%
		CPU	3.30ms	0.56ms (17.05%)	0.18ms (5.39%)	-25.14%
wyc8b	no	real	4.16ms	0.69ms (16.56%)	0.22ms (5.24%)	0.00%
		CPU	4.06ms	0.47ms (11.53%)	0.15ms (3.65%)	0.00%
	yes	real	3.42ms	0.61ms (17.77%)	0.19ms (5.62%)	-17.74%
		CPU	3.31ms	0.59ms (17.75%)	0.19ms (5.61%)	-18.40%
wyc8c	no	real	8.76ms	0.91ms (10.43%)	0.29ms (3.30%)	0.00%
		CPU	8.08ms	0.61ms (7.60%)	0.19ms (2.40%)	0.00%
	yes	real	7.53ms	0.84ms (11.20%)	0.27ms (3.54%)	-14.04%
		CPU	7.31ms	0.71ms (9.70%)	0.22ms (3.07%)	-9.62%
wyc8d	no	real	5.72ms	0.97ms (16.92%)	0.31ms (5.35%)	0.00%
		CPU	5.47ms	0.83ms (15.12%)	0.26ms (4.78%)	0.00%
	yes	real	5.59ms	0.69ms (12.32%)	0.22ms (3.90%)	-2.21%
		CPU	5.46ms	0.70ms (12.74%)	0.22ms (4.03%)	-0.27%
wyc8e	no	real	6.55ms	1.15ms (17.63%)	0.37ms (5.57%)	0.00%
		CPU	6.26ms	0.87ms (13.83%)	0.27ms (4.37%)	0.00%
	yes	real	5.57ms	0.77ms (13.74%)	0.24ms (4.34%)	-14.98%
		CPU	5.33ms	0.73ms (13.80%)	0.23ms (4.36%)	-14.96%
wyc9a	no	real	10.58ms	0.74ms (7.01%)	0.23ms (2.22%)	0.00%
		CPU	10.37ms	0.66ms (6.41%)	0.21ms (2.03%)	0.00%
	yes	real	9.81ms	0.84ms (8.61%)	0.27ms (2.72%)	-7.24%
		CPU	9.60ms	0.83ms (8.65%)	0.26ms (2.74%)	-7.49%

wyc9b	no	real	2.42ms	0.58ms (24.13%)	0.18ms (7.63%)	0.00%
		CPU	2.46ms	0.59ms (24.08%)	0.19ms (7.61%)	0.00%
	yes	real	1.95ms	0.40ms (20.34%)	0.13ms (6.43%)	-19.35%
		CPU	1.82ms	0.36ms (19.79%)	0.11ms (6.26%)	-26.02%
wyc9c	no	real	20.95ms	1.85ms (8.83%)	0.58ms (2.79%)	0.00%
		CPU	20.31ms	1.77ms (8.73%)	0.56ms (2.76%)	0.00%
	yes	real	19.92ms	1.10ms (5.54%)	0.35ms (1.75%)	-4.91%
		CPU	19.68ms	1.11ms (5.62%)	0.35ms (1.78%)	-3.07%
wyc9d	no	real	7.17ms	0.50ms (7.00%)	0.16ms (2.21%)	0.00%
		CPU	7.20ms	0.49ms (6.84%)	0.16ms (2.16%)	0.00%
	yes	real	7.59ms	0.83ms (10.92%)	0.26ms (3.45%)	5.88%
		CPU	7.42ms	0.80ms (10.84%)	0.25ms (3.43%)	3.10%
wyc9e	no	real	18.75ms	2.14ms (11.43%)	0.68ms (3.62%)	0.00%
		CPU	17.90ms	1.39ms (7.79%)	0.44ms (2.46%)	0.00%
	yes	real	16.00ms	0.05ms (0.29%)	0.01ms (0.09%)	-14.67%
		CPU	15.80ms	0.02ms (0.11%)	0.01ms (0.03%)	-11.71%
wyc9f	no	real	14.04ms	0.67ms (4.76%)	0.21ms (1.51%)	0.00%
		CPU	14.04ms	0.66ms (4.73%)	0.21ms (1.50%)	0.00%
	yes	real	13.40ms	0.04ms (0.30%)	0.01ms (0.10%)	-4.60%
		CPU	13.20ms	0.02ms (0.12%)	0.00ms (0.04%)	-5.99%
wyc10a	no	real	56.45ms	0.07ms (0.12%)	0.02ms (0.04%)	0.00%
		CPU	56.36ms	0.06ms (0.11%)	0.02ms (0.03%)	0.00%
	yes	real	56.33ms	0.06ms (0.10%)	0.02ms (0.03%)	-0.20%
		CPU	56.05ms	0.05ms (0.09%)	0.02ms (0.03%)	-0.55%
wyc10b	no	real	4.48ms	0.07ms (1.67%)	0.02ms (0.53%)	0.00%
		CPU	4.53ms	0.07ms (1.59%)	0.02ms (0.50%)	0.00%
	yes	real	4.08ms	0.02ms (0.47%)	0.01ms (0.15%)	-8.97%
		CPU	3.96ms	0.02ms (0.49%)	0.01ms (0.15%)	-12.52%
wyc10c	no	real	141.29ms	5.77ms (4.08%)	1.82ms (1.29%)	0.00%
		CPU	139.76ms	2.69ms (1.93%)	0.85ms (0.61%)	0.00%
	yes	real	139.08ms	0.20ms (0.15%)	0.06ms (0.05%)	-1.56%
		CPU	138.57ms	0.20ms (0.14%)	0.06ms (0.04%)	-0.86%
wyc10d	no	real	162.04ms	13.65ms (8.42%)	4.32ms (2.66%)	0.00%
		CPU	158.48ms	11.28ms (7.12%)	3.57ms (2.25%)	0.00%
	yes	real	171.33ms	3.61ms (2.11%)	1.14ms (0.67%)	5.73%
		CPU	168.48ms	1.68ms (1.00%)	0.53ms (0.32%)	6.31%
wyc10e	no	real	121.74ms	7.23ms (5.94%)	2.29ms (1.88%)	0.00%
		CPU	118.23ms	6.55ms (5.54%)	2.07ms (1.75%)	0.00%
	yes	real	105.10ms	0.56ms (0.54%)	0.18ms (0.17%)	-13.67%
		CPU	104.67ms	0.57ms (0.54%)	0.18ms (0.17%)	-11.47%
wyc11a	no	real	95.76ms	0.08ms (0.09%)	0.03ms (0.03%)	0.00%
		CPU	95.61ms	0.07ms (0.08%)	0.02ms (0.02%)	0.00%
	yes	real	95.17ms	0.10ms (0.11%)	0.03ms (0.03%)	-0.62%
		CPU	94.80ms	0.09ms (0.09%)	0.03ms (0.03%)	-0.86%
wyc11b	no	real	129.05ms	10.32ms (8.00%)	3.26ms (2.53%)	0.00%
		CPU	126.86ms	8.86ms (6.99%)	2.80ms (2.21%)	0.00%
	yes	real	133.16ms	3.21ms (2.41%)	1.02ms (0.76%)	3.19%
		CPU	131.28ms	2.79ms (2.13%)	0.88ms (0.67%)	3.49%
wyc11c	no	real	232.34ms	17.40ms (7.49%)	5.50ms (2.37%)	0.00%
		CPU	230.35ms	15.38ms (6.67%)	4.86ms (2.11%)	0.00%
	yes	real	230.02ms	15.13ms (6.58%)	4.78ms (2.08%)	-1.00%
		CPU	229.20ms	15.02ms (6.55%)	4.75ms (2.07%)	-0.50%

wyc11d	no	real	298.05ms	6.90ms (2.32%)	2.18ms (0.73%)	0.00%
		CPU	295.28ms	3.71ms (1.26%)	1.17ms (0.40%)	0.00%
	yes	real	297.98ms	5.52ms (1.85%)	1.74ms (0.59%)	-0.02%
		CPU	295.75ms	3.38ms (1.14%)	1.07ms (0.36%)	0.16%
wyc11e	no	real	246.66ms	17.21ms (6.98%)	5.44ms (2.21%)	0.00%
		CPU	246.24ms	17.21ms (6.99%)	5.44ms (2.21%)	0.00%
	yes	real	232.43ms	17.78ms (7.65%)	5.62ms (2.42%)	-5.77%
		CPU	229.53ms	11.28ms (4.91%)	3.57ms (1.55%)	-6.79%
wyc12a	no	real	167.16ms	5.64ms (3.38%)	1.78ms (1.07%)	0.00%
		CPU	162.40ms	1.48ms (0.91%)	0.47ms (0.29%)	0.00%
	yes	real	162.51ms	1.71ms (1.05%)	0.54ms (0.33%)	-2.78%
		CPU	161.87ms	1.70ms (1.05%)	0.54ms (0.33%)	-0.32%
wyc12b	no	real	49.68ms	4.88ms (9.82%)	1.54ms (3.10%)	0.00%
		CPU	47.40ms	2.65ms (5.59%)	0.84ms (1.77%)	0.00%
	yes	real	48.67ms	3.08ms (6.33%)	0.97ms (2.00%)	-2.03%
		CPU	47.87ms	2.60ms (5.43%)	0.82ms (1.72%)	1.00%
wyc12c	no	real	365.87ms	27.67ms (7.56%)	8.75ms (2.39%)	0.00%
		CPU	363.64ms	26.97ms (7.42%)	8.53ms (2.35%)	0.00%
	yes	real	389.56ms	26.92ms (6.91%)	8.51ms (2.19%)	6.48%
		CPU	385.19ms	22.96ms (5.96%)	7.26ms (1.89%)	5.93%
wyc12d	no	real	368.08ms	15.34ms (4.17%)	4.85ms (1.32%)	0.00%
		CPU	366.39ms	12.87ms (3.51%)	4.07ms (1.11%)	0.00%
	yes	real	422.77ms	9.76ms (2.31%)	3.09ms (0.73%)	14.86%
		CPU	415.08ms	4.78ms (1.15%)	1.51ms (0.36%)	13.29%
wyc12e	no	real	395.75ms	30.44ms (7.69%)	9.63ms (2.43%)	0.00%
		CPU	389.24ms	25.85ms (6.64%)	8.17ms (2.10%)	0.00%
	yes	real	361.03ms	13.54ms (3.75%)	4.28ms (1.19%)	-8.77%
		CPU	357.38ms	8.59ms (2.40%)	2.72ms (0.76%)	-8.18%
wyc12f	no	real	321.16ms	26.00ms (8.09%)	8.22ms (2.56%)	0.00%
		CPU	316.60ms	22.61ms (7.14%)	7.15ms (2.26%)	0.00%
	yes	real	344.04ms	1.45ms (0.42%)	0.46ms (0.13%)	7.12%
		CPU	342.77ms	1.42ms (0.42%)	0.45ms (0.13%)	8.27%

Table A.8: Run times of the model solution program of the problem Wycieczki (wyc) from the finale of XXII Polish Olimpiad in Informatics. For each configuration (Test case and Sandbox columns) the data was collected from 10 runs. Real and CPU times were collected from the same runs. Slowdown is measured from the times of configuration without the sandbox.