WCRT for Synchronous Programs: Studying the Tick Alignment Problem

Michael Mendler, Bruno Bodin, Partha S. Roop, Jia Jie Wang
Worst-case Reaction Time for Synchronous Programs: Studying the Tick Alignment Problem

Michael Mendler∗
Bamberg University
Germany
michael.mendler@uni-bamberg.de

Bruno Bodin
University of Edinburgh
United Kingdom
bbodin@inf.ed.ac.uk

Partha S Roop
University of Auckland
New Zealand
p.roop@auckland.ac.nz

Jia Jie Wang
University of Auckland
New Zealand
jwan232@aucklanduni.ac.nz

Abstract

Synchronous programs are ideally suited for the design of safety critical systems as they provide guarantees on determinism and deadlock freedom. In addition to such functional guarantees, guarantees on timing are also necessary. In this report, we study the problem of static worst case reaction time (WCRT) analysis of synchronous programs.

While, there have been many recent attempts at studying this problem from the point of view of scalability and precision, one crucial aspect is yet to be examined from a fundamental viewpoint. Concurrent threads in a synchronous programs must align during every reaction, a problem that has been termed as the tick alignment problem (TAP), i.e., infeasible ticks that never align in practice must be ruled out for precision. We, for the first time, study TAP in the guise of a number theoretic formulation in order to not only explore its lower bound complexity, but also to develop heuristics that work well in practice. The developed algorithm that is based on the Maximum Weight Clique Problem. Extensive benchmarking reveals the relative superiority of the proposed approach. While being optimal it is also more efficient compared to one of the most efficient of known techniques, ILP\textsubscript{C}, which uses iterative approximation with integer linear programming techniques. Finally, using insights from the proposed TAP formulation, we develop a refinement of ILP\textsubscript{C}, called ILP\textsubscript{CP}, that excels in comparison to all known techniques for WCRT analysis.

1 Introduction

The synchronous paradigm [3] is ideal for designing safety critical systems in aviation, automotive and industrial automation. Synchronous languages offer a simple mechanism, based on a logical global clock, for thread synchronization. This removes the inter-leavings and associated non-determinism of asynchronous composition, resulting in a framework that is more amenable for static analysis for functional correctness. The issue of timing correctness is at the heart of many real-time safety critical systems and is the topic of our interest. Considering the simplicity of synchronous composition, an obvious conjecture would be that timing correctness of synchronous programs should also be simpler

∗The author is supported by a grant from the German Science Foundation (DFG ME 1427/6-1).
in comparison. Is this conjecture valid? This aspect is the central theme of the current investigation.

Timing correctness of synchronous programs is closely intertwined with that of the *synchrony hypothesis*, which asserts that the synchronous program operates infinitely fast relative to its environment. Practical implementations validate this by ensuring that inputs from the environment never happen at a rate that is faster than the *worst case reaction time* (WCRT) of any synchronous reaction (also known as *tick*). Compared to the problem of worst case execution time (WCET) [16] of sequential programs, WCRT analysis has received much less attention. However, interest in this topic has been growing, with many recent attempts that primarily explore the trade-off between precision and analysis time. We may broadly classify these under the following categories:

1. Maximum thread cost [4, 10]: These approaches compute the maximum tick lengths for every thread (termed their local ticks) and then compute the sum of these maximum local ticks to determine the WCRT. These, while being the fastest known approach, have been shown to produce large overestimates.

2. Implicit path enumeration [7, 8]: These approaches rely on integer linear programming (ILP) to model the flow constraints of a control flow graph and are inspired by traditional ILP-based techniques for WCET analysis of sequential programs [16]. Hence, they first convert the concurrent control flow of the synchronous program into its sequential equivalent before applying the ILP formulation. They can be used for pruning infeasible paths to obtain very precise WCRT values. However, have a higher complexity (NP hard) compared to the polynomial complexity of approach-1. We call this approach as ILP<sub>S</sub> (ILP sequential).

3. State exploration [9, 18]: These approaches work directly on the concurrent control flow to accurately compute the worst case tick length by examining all possible synchronous thread inter-leavings that are valid. These approaches compute precise WCRT value at the expense of exponential worst case complexity. In a recent paper, Wang et al. [15] have compared model checking [14], reachability [9], and ILP<sub>S</sub> [7]. This shows that reachability works best in practice compared to the other techniques for large state space (10 million states and beyond).

4. Iterative tightening [15, 13]: Wang et al. [15] noticed that there is a trade-off between approach-1 and the other approaches based on either path enumeration (approach-2) or state exploration (approach-3). They have developed an iterative refinement based approach called ILP<sub>C</sub> (ILP concurrent), by the creation of two different ILP models on the concurrent control flow graph. The first model is used to compute an over-approximation using the maximum cost of local ticks and uses a second ILP model to check if the computed over-approximation is infeasible (i.e., the associated ticks won’t align during execution). They iteratively refine this until the most precise value is computed. They have shown that ILP<sub>C</sub> performs the best among known approaches for large benchmarks. Independently from this, a strategy for iterative tightening has been proposed by Raymond et al. [13] for the ILP<sub>S</sub> method (approach-2). They employ flow facts or infeasibility properties (verified as invariants using a model-checker) at the high-level source language (Lustre) to derive low-level path constraints on the scheduled sequential program to guide the ILP solver towards tighter WCRT values.

These recent attempts at solving the WCRT problem have been guided mainly by practical considerations. Existing general-purpose analysis algorithms (ILP, model-checking,
SAT-solving, micro-architectural modelling tools) are applied to different synchronous languages and (precision-timed) hardware architectures. Investigations of the WCRT problem from a language- and architecture-independent perspective are rare. Because of the inherent complexity of the problem it seems almost unavoidable that efficiency, scalability and precision can only be measured using benchmarks rather than analytical methods. However, it seems clear too, that if the WCRT problem is only studied in highly specific, incompatible engineering contexts, the results on WCRT analysis methods become scientifically meaningless. Yet, as the importance of WCRT analysis is increasingly recognised and the synchronous programming methodologies become more varied, it is essential to improve our understanding of the fundamental logical and algorithmic nature of the WCRT problem. This is a prerequisite to obtain a sound basis for terminological distinctions characterising the different practical algorithms and to carry over the results on WCRT analysis from one practical scenario to another. How can we obtain universal benchmark suites to compare our results? What do we know about the algorithmic complexity of the WCRT problem?

In this paper we consider one fundamental and language-independent aspect of the WCRT problem that has been termed as the tick alignment problem (TAP). Concurrent threads in a synchronous program jointly switch from one tick to the next and thus must align their reactions. Infeasible ticks that never align must be ruled out for precision. Even if the timing is independent of the communication, and the duration of each tick in each thread is known, calculating the exact timing of the synchronous composition is non-trivial. Experimental evidence from recent papers such as [15] indicate that the worst case complexity of this problem may be exponential. However, the lower-bound complexity of TAP has neither been studied nor established. Our work aims to close this gap. The main contributions reported here are:

1. We provide a polynomial transformation from an intermediate representation of synchronous programs called a tick cost automaton (TCA) to an equivalent monocyclic form (m-TCA) using max-plus power series.

2. We show how this transformation facilitates the development of a number theoretic formulation to solve TAP based on solving linear congruences using the well-known Chinese Remainder Theorem.

3. Based on the number-theoretic analysis we are able to study TAP as a graph-theoretic problem with a view to understanding its lower bound complexity. More specifically, we relate TAP to the well-known maximal (weighted) clique problem [17, 11].

4. We strengthen one of the most efficient WCRT analysis techniques based on iterative narrowing, called ILP$_C$, by pairwise compatibility checking introduced here. The proposed algorithm, called ILP$_{CP}$ is shown through extensive benchmarking to be the best known method for solving TAP.

5. Our experiments show that heuristic polynomial algorithms from the number and graph theoretic specification can compute precise WCRT values for many practical TAP benchmarks.

Our results suggest that TAP is an interesting candidate for a universal WCRT problem, as it seems to be practically solvable in polynomial time but whose lower-bound computational complexity, as far as we are aware, is still unknown.
1.1 Overview and organization

We assume that synchronous programs can be converted to a set of finite automata, which we term as tick cost automata (defined in Section 2). Such a transformation is feasible and used by earlier approaches such as [14, 1]. Given a set of TCAs that operate synchronously, we propose a transformation to convert them to a monocyclic form in Section 2 derived from an algebraic coding as max-plus power series. Following this, we present a number-theoretic formulation of TAP in Section 4. We present an algorithm for solving the TAP problem in Section 5 followed by benchmarking in Section 6. The report is concluded in the final Section 7.

2 Tick Cost Automata and the Tick Alignment Problem

The timing behaviour of a synchronous program and its parts is modelled by a tick cost automaton, or TCA for short. A TCA is an abstract representation of a sequential (single-threaded) synchronous program in which all data dependencies have been abstracted away while the control-flow structure is preserved. Computing the WCRT of such a TCA yields an over-approximation of the WCRT of the underlying synchronous program which ignores the potential infeasibility of flow paths arising from conditional tests on data. Solutions for this intra-thread sensitization problem are well-known and not treated here. Instead, our focus is on the inter-thread sensitisation problem that arises from the synchronous coupling of ticks across concurrent TCAs.

Definition 2.1 A tick cost automaton (TCA) is a tuple $A = (Q, \rightarrow, e, F)$, where $Q$ is a finite set of states partitioned into the set of transient states and pause states, $Q = Q_t \uplus Q_p$. The distinguished entry state $e \in Q_t$ and all the exit states $F \subseteq Q_t$ are transient. The transition relation $\rightarrow \subseteq Q \times \mathbb{N} \times Q$ is labelled by natural numbers and we write $d : q_1 \rightarrow q_2$ for $(q_1, d, q_2) \in \rightarrow$.

A TCA is a finite state automaton where the states model the control points of the program and the transitions the possible executions paths between them, labelled by the time needed to reach one control point from the other. A transient state is a control point which, when entered, is instantaneously left in the same tick. In contrast, when the control flow reaches a pause state it pauses and waits for the next synchronous clock tick. Following the terminology of [10] we can distinguish four types of finite and complete execution paths in a TCA:

- A through path is any sequence of transient states connected by transitions, starting in the entry state $e$ and ending in some exit state $f \in F$. These correspond to computations in which the TCA is entered and exited instantaneously within the same tick.

- A sink path starts in $e$, passes through an arbitrary number of transient states and then ends in a pause state $f \in Q_p$. On a sink path a synchronous tick enters the TCA and then pauses inside it.

- An internal path begins and ends in a pause state, while all intermediate states are transient. These paths capture the normal synchronous operation where control fully resides in the TCA during a tick.

- A source path begins in a pause state but then only visits transient states until it ends in an exit state. These correspond to executions in which the TCA is active at the beginning but then instantaneously left during the tick.
The through and sink paths together correspond to executions of the so-called *surface behaviour* of the TCA, i.e., instantaneous executions entering the TCA until they either reach a first pause or exit the TCA. The internal and source paths constitute the so-called *depth behaviour* of the TCA.

The timing of a TCA is captured in the labels of the transitions. These express an instantaneous duration quantified by natural numbers $\mathbb{N}$ and counting low-level instruction cycles of a processor or some other operationally meaningful physical unit of time. Note that since we abstract from the data communications of a synchronous program, TCAs are non-deterministic and the transition times are safe over-approximations of the exact execution time which may depend on the environment input or other low-level parameters not modelled by the TCAs.

![Diagram of TCA](image)

**Figure 1:** A TCA $A$ (left) and its equivalent linear reduced form $A^*$ (right) (see Sec. 3.2).

An example of a TCA is shown in Fig. 1 on the left which represents the abstract tick automaton of a synchronous program with time annotations to describe the maximal duration of synchronous instants. This automaton $A$ has transient states $A_0$, $A_5$ and $A_6$ drawn as solid circles, and pause states $A_1$, $A_2$, $A_3$ and $A_4$ drawn as two half-circles. The transient entry node $A_6$ is indicated by a transition arrow without source state. There are no exit states in this automaton.

Each pause state is split into two parts. The upper half of each pause state represents the *surface* of the state which, when reached, causes the end of the instant. The control flow pauses there to wait for the clock tick. The occurrence of the clock tick switches activation from $\text{tick}(A_i)$ to the lower half of the state, called the *depth*, from where the successive instant then is started. To express the synchronising behaviour of the clock tick we always use $\text{tick}(q)$ for the surface and $q$ for the depth of a pause state in a TCA. This is indicated only for state $A_2$ in Fig. 1 but applies to all other pause states, too.

Any internal path starts the automaton in some pause state $A_i$ (the depth part) at the beginning of the tick, then activates a sequence of transitions through transient states and finally pauses in $\text{tick}(A_j)$ of a successor state $A_j$ (the surface part). For instance, in $A$ an instant might start in $A_2$ and end in $\text{tick}(A_3)$ with a maximal duration of 31 time units, or end in $\text{tick}(A_4)$ after maximal 32 time units. An example of a sink path for $A$ of Fig. 1 begins in $A_6$ and ends in $\text{tick}(A_1)$ or in $\text{tick}(A_2)$. As can be seen the TCA $A$ has no through paths and no source paths. This means, whenever the execution of an instant enters $A$ (though $A_6$) it remains inside $A$ for the current and all successive ticks. It is important to keep in mind that the choices for sink or internal paths is merely...
a non-determinism of modelling not a non-determinism of execution. It is resolved at run time by the actual synchronous program whose timing behaviour is modelled by $A$. The non-determinism arises naturally in the compositional translation from synchronous programs to TCAs (see [14, 1]) as soon as we abstract from data and signal dependencies. We shall see below in Sec. 3.2 how the non-determinism can be eliminated by reduction to an equivalent linear reduced form. For our example TCA $A$ this deterministic and reduced TCA $B$ is seen on the right of Fig. 1.

Note that a general TCA can contain transient cycles, i.e., loops in which all states are transient. However, synchronous programs are usually verified to be constructive, which implies they do not have (executable) instantaneous cycles. Hence, we may assume in this report that each cycle in a TCA contains at least one pause state.

Another simplification that we will make for the present purposes is to assume that a TCA has neither source nor through paths, like the example automaton in Fig. 1. This is the same as saying that there are no exit states, i.e. if $A = \langle Q, \rightarrow, e, F \rangle$ then $F = \emptyset$. Of course, exit states are important for sequential composition of TCAs. However, as we will only be concerned with parallel composition in this report, we can do without them. So, the surface behaviour of the TCAs considered here consists of paths from the entry state to the first pause state and the depth behaviour consists of paths between pause states.

Let us now look at how a TCA models a WCRT problem. In the timing analysis of a synchronous tick the transition delays must be added up along transient paths through the TCA. Let us write $d : q_0 \rightarrow q_k$, for $k \geq 1$, if there exists a (possibly empty) sequence of transient states $q_1, q_2, \ldots, q_{k-1} \in Q_t$ such that $d_1 : q_0 \rightarrow q_1, d_2 : q_1 \rightarrow q_2, \ldots, d_k : q_{k-1} \rightarrow q_k$, and $d = d_1 + d_2 + \cdots + d_k$.

**Definition 2.2** The worst case reaction time of a TCA $A$ is

$$\text{wcrt}(A) \stackrel{df}{=} \max \{d \mid d : q_0 \rightarrow q_k, q_0, q_k \in Q\},$$

which returns the maximum cost of any transient path in $A$.

Note that our definition of $\text{wcrt}(A)$, which refers to arbitrary start and end states $q_0, q_k \in Q$, is somewhat more general than the traditional definition of synchronous WCRT which is taken as the maximum delay between any two pause states, i.e., the maximal length of an internal path. Our definition of $\text{wcrt}(A)$ measures all the transient parts of $A$, too. Specifically, it also covers the time it takes to reach the first pause state from the entry state of $A$ (sink paths), the time from any pause state to reach an exit of $A$ (source paths), and the instantaneous delay from the entry of $A$ to an exit that does not reach any pause state (through). We have chosen Def. 2.2 to match the generality of the notion of a TCA in Def. 2.1, although we will later essentially talk about internal paths only.

In the example $A$ of Fig. 1 (left) the maximal cost transient path is $A4 \rightarrow A5 \rightarrow \text{tick}(A1)$
of weight 36, so that $\text{wcrt}(A) = 36$. It is easy to compute $\text{wcrt}(A)$ for any TCA, in polynomial time, by dynamic programming techniques, e.g., using a suitable modification of Floyd-Warshall’s all-pairs shortest path algorithm [5].

The WCRT problem becomes computationally interesting when we look at a parallel composition of TCAs. In a synchronous multi-threaded composition $A \parallel B$, the two TCAs

\footnote{This is an over-simplification in the sense that the abstraction from data may force a TCA $A$ to have instantaneous cycles in order to remain finite. However, on a transient cycle the WCRT is either 0 or $\infty$, so that the cycle can either be ignored, or the timing analysis stops with $\text{wcrt}(A) = \infty$.}
A and B run concurrently by interleaving their transitions. They synchronise their ticks so that A∥B reaches a pause state whenever both A and B reach a pause state. As soon as one component reaches a pause state it stops and waits for the other to reach a pause.

**Definition 2.3** For any two TCAs \( A = (Q^A, \rightarrow^A, e^A) \) and \( B = (Q^B, \rightarrow^B, e^B) \) their synchronous multi-threaded product \( A\parallel B = (Q, \rightarrow, e) \) is given as follows: The set of pause and transient states are given as

- \( Q \overset{df}{=} Q_t \sqcup Q_p \)
- \( Q_p \overset{df}{=} Q^A_p \times Q^B_p \)
- \( Q_t \overset{df}{=} (Q^A_t \times Q^B_t) \cup (\text{tick}(Q^A_p) \times Q^B_t) \cup (Q^A_t \times \text{tick}(Q^B_p)) \),

where \( \text{tick}(Q) \overset{df}{=} \{ \text{tick}(q) \mid q \in Q \} \). The entry state is \( e \overset{df}{=} (e^A, e^B) \) and the transition relation \( \rightarrow \) is the least relation closed under the following rules:

\[
\begin{align*}
q^B &\in Q^B & q^A &\in Q^A & r^A &\in Q^A_t \quad d : q^A \rightarrow^A e^A_u & \quad (\text{asy}_1) \\
\text{tick}(q^B) &\in Q^B & r^A &\in Q^A_t \quad d : (q^A, q^B) \rightarrow (r^A, q^B) & \quad (\text{asy}_2) \\
q^A &\in Q^A & q^B &\in Q^B & r^B &\in Q^B_t \quad d : q^B \rightarrow^B r^B & \quad (\text{asy}_2) \\
\text{tick}(q^A) &\in Q^A & r^A &\in Q^A_t \quad d : (q^A, q^B) \rightarrow (q^A, r^B) & \quad (\text{asy}_2) \\
p^B &\in Q^B & q^A &\in Q^A & p^A &\in Q^A_p \quad d_1 : q^A \rightarrow^A p^A & \quad (\text{syn}_1) \\
p^A &\in Q^A & q^B &\in Q^B & p^B &\in Q^B_p \quad d_1 : (q^A, \text{tick}(p^B)) \rightarrow (p^A, p^B) & \quad (\text{syn}_2) \\
p^A &\in Q^A & q^B &\in Q^B & p^B &\in Q^B_p \quad d_2 : q^B \rightarrow^B p^B & \quad (\text{syn}_2) \\
\text{tick}(q^A) &\in Q^A & p^A &\in Q^A_p \quad d : (\text{tick}(p^A), q^B) \rightarrow (p^A, p^B) & \quad (\text{syn}_2) \\
q^B &\in Q^B & q^A &\in Q^A & p^A &\in Q^A_p \quad d_1 : q^A \rightarrow^A p^A & \quad (\text{syn}_3) \\
p^A &\in Q^A & q^B &\in Q^B & p^B &\in Q^B_p \quad d_1 : (q^A, q^B) \rightarrow (q^A, \text{tick}(p^B)) & \quad (\text{syn}_3) \\
q^A &\in Q^A & q^B &\in Q^B & p^B &\in Q^B_p \quad d_2 : q^B \rightarrow^B p^B & \quad (\text{syn}_3) \\
\text{tick}(q^A) &\in Q^A & p^A &\in Q^A_p \quad d : (\text{tick}(p^A), q^B) \rightarrow (q^A, \text{tick}(p^B)) & \quad (\text{syn}_3)
\end{align*}
\]

Def. 2.3 models a synchronised interleaving of two TCAs. The interleaving can be seen from the fact that each transition of \( C\parallel D \) (and the associated delay) stems from exactly one transition of the component TCAs.

As an example consider the parallel composition \( C\parallel D \) shown in Fig. 2. The entry state of \( C\parallel D \) is \( (C^0, D^0) \) which is transient. From there either one of the TCAs can make a move, specifically \( 5 : (C^0, D^0) \rightarrow (\text{tick}(C^1), D^0) \) or \( 1 : (C^0, D^0) \rightarrow (C^0, \text{tick}(D^1)) \), by rules asy\_1 or asy\_2, respectively. Both successor states \( (\text{tick}(C^1), D^0) \) and \( (C^0, \text{tick}(D^1)) \) are still transient. The tick prefix in \( \text{tick}(C^1) \) and \( \text{tick}(D^1) \) indicates that the TCA which has moved has reached a pause state and now is waiting for the other to finish the tick. E.g., in \( (\text{tick}(C^1), D^0) \) the automaton C waits in its pause state C1 while D is still in the transient entry state D0. As soon as D moves into it pause state D1, too, we reach the global state \( (C^1, D_1) \). This happens in the transition \( 1 : (\text{tick}(C^1), D^0) \rightarrow (C^1, D^1) \) generated by rule syn\_2. Notice that the tick prefix is removed, making \( (C^1, D^1) \) a pause state of the composite TCA \( C\parallel D \).
Figure 2: A synchronous product of TCAs $C$ and $D$ with aligned pause states (left) and the composite TCA $C\|D$ (right).

The operation $\|/A$ is associative and commutative in the sense that $(A\|B)/C$ and $A\|(B/\|C)$, and $A\|B$ and $B\|/A$ respectively, are isomorphic TCAs, i.e., the same automaton modulo the representation of their states. Therefore, we can use the operator $\|$ as a multi-ary combinator without need to write brackets.

Our Def. 2.3 captures the timing behaviour of synchronous composition in the sense that if $A_P$ and $A_Q$ are abstractions of synchronous programs $P$ and $Q$, respectively, then $A_P\|/A_Q$ is sound with respect to the executions of the tick-synchronized multi-threaded composition $P\|/Q$ of $P$ and $Q$. However, just like $A_P$ and $A_Q$ are abstractions of the execution semantics of $P$ and $Q$, the composition $A_P\|/A_Q$ of their TCAs, in general, is a sound over-approximation of the control paths executable by $P\|/Q$. In particular, $\text{wcr}(A_P\|/A_Q)$ cannot be precise because $P$ and $Q$ communicate through signal values, which introduces additional scheduling constraints that would need the tracking of signal data in $A_P$, $A_Q$ and in Def. 2.3, which we ignore. We will see that even under this drastic simplification of parallel composition both the WCRT problem itself as well as the achievable precision of the WCRT results are non-trivial.

**Definition 2.4** The Tick Alignment Optimisation Problem TAP is the problem to compute $\text{wcrt}(T)$ for an arbitrary parallel composition $T = T_1\|/T_2\|\cdots\|/T_n$ of TCAs $T_i$ with $i = 1,\ldots,n$.

Let us have a look at our example $C\|/D$ from Fig. 2. By inspecting the transition system for $C\|/D$ on the right of Fig. 2 we can work out that the longest transient execution path has duration

$$\text{wcrt}(C\|/D) = 16 = 13 + 3 = 3 + 13 : (C2, D2) \rightarrow (C3, D1).$$
To understand the tick alignment problem it is important to observe that the worst case reaction time $\text{wcrt}(C \parallel D)$ is smaller than the sum of the WCRT of the two component processes which is $\text{wcrt}(C) + \text{wcrt}(D) = 13 + 12 = 26$. The reason is that the two pause states from which these worst case behaviours are observable, viz. $C2$ and $D1$, never come to be active in the same tick. They are not tick aligned. The pause states which are tick aligned are indicated by the dotted lines connecting $C$ and $D$ in Fig. 2. These are the pairs of states that appear as reachable pause states in the composite TCA, viz. $\{(C1, D1), (C2, D2), (C3, D1), (C4, D2)\}$. The pause state $(C2, D1)$ which is included as a pause state in the generic Def. 2.3 of $C \parallel D$ is not reachable.

3 Max-Plus Semantics of TCA

In order to solve the Tick Alignment Problem it is useful to study the algebraic manipulations on TCAs based on a notion of equivalence that preserves the WCRT semantics of the automata. For our purposes the canonical definition is to stipulate $A \cong B$ iff both $A$ and $B$ generate the same WCRT for all parallel contexts $C$, i.e., for all TCAs $C$ we have $\text{wcrt}(A || C) = \text{wcrt}(B || C)$. Trivially, this yields an equivalence relation such that $A \cong B$ implies $\text{wcrt}(A) = \text{wcrt}(B)$ and which is a congruence for parallel composition, i.e., if $A \cong B$ then $A || C \cong B || C$ in all parallel contexts $C$. The practical importance of the equivalence $\cong$ is that it turns TCAs into an algebra and permits us to reduce the problem of computing $\text{wcrt}(A \parallel B)$ to the problem of computing $\text{wcrt}(A^* || B^*)$ where $A^*$ and $B^*$ are simplified versions of $A$ and $B$ with $A \cong A^*$ and $B \cong B^*$. The next step is to identify an expressively complete set of operators on TCAs and $\cong$-simplification rules for TCAs using these operators.

3.1 Formal Max-Plus Power Series

Specifically, it turns out that TCAs correspond to formal power series in the max-plus algebra $(\mathbb{N}_\infty, \oplus, \odot, 0, 1)$ where $\mathbb{N}_\infty \overset{df}{=} \mathbb{N} \cup \{ -\infty \}$ and $\oplus$ stands for the maximum and $\odot$ for addition on $\mathbb{N}_\infty$. Both binary operators $\oplus$ and $\odot$ are commutative, associative and have the neutral elements $0 \overset{df}{=} -\infty$ and $1 \overset{df}{=} 0$, respectively, i.e., $x \oplus 0 = x$ and $x \odot 1 = x$. The constant $0$ is absorbing for $\odot$, i.e., $x \odot 0 = 0 \odot x = 0$. Finally, $\odot$ distributes over $\oplus$, i.e., $x \odot (y \oplus z) = (x \odot y) \oplus (x \odot y)$ which is the same as $x + \max(y, z) = \max(x + y, x + z)$. However, $\oplus$ does not distribute over $\odot$, for instance, $4 \oplus (5 \odot 2) = \max(4, 5 + 2) = 7$ while $(4 \odot 5) \oplus (4 \odot 2) = \max(4, 5) + \max(4, 2) = 9$. This explains the choice of notation $\odot$ and $\oplus$ to highlight the multiplicative and additive nature, respectively, of the operators.\(^2\)

As in standard arithmetic we write multiplicative expressions $x \odot y$ also without the operator simply as $xy$. A comprehensive study of the theory of max-plus algebra, and its generalisation, the dioids, can be found in [2]. The important role of this structure for solving path problems is highlighted also in [5], where it is called a semiring.

The structure $\mathbb{N}_\infty$ plays the role of scalars in our algebra of TCAs where each TCA is identified, up to $\cong$, with a formal power series, or fps for short,

$$F[X] = F_0 \oplus F_1 X \oplus F_2 X^2 \oplus F_3 X^3 \cdots \quad (1)$$

with scalars $F_i \in \mathbb{N}_\infty$ and where exponentiation is repeated multiplication, i.e., $X^0 = 1$ and $X^{k+1} = X X^k = X \odot X^k$. Such a fps (1) stores an infinite sequence of numbers $F_0, F_1, F_2, F_3, \ldots$ as the coefficients of the base polynomials $X^k$. In contrast to normal power series a formal power series does not need to converge as a function of variable $X$.

\(^2\)We are grateful to Alain Girault who pointed out to us the rather natural notation for the constants $0$ and $1$. 

9
For instance, considering that \( 1^k = 1 \odot 1 \odot \cdots \odot 1 = k \) we have \( F[1] = F_0 \oplus F_1 \odot 1 \oplus F_2 \odot 2 \oplus F_3 \odot 3 \odot \cdots \) which diverges to \( \infty \), unless all but a finite number of coefficients \( F_k \) are \( 0 \). Also, \( F[1] = F_0 \oplus F_1 \odot 1 \oplus F_2 \odot 2 \oplus \cdots \) only converges if the number series \( F_0, F_1, F_2, \ldots \) is bounded. In this case \( F[1] \) is the maximum of all the coefficients. All the sequences considered in this report, which are generated by finite-state TCAs, are bounded. Hence, \( F[1] \) is always defined. When \( A[X] \) codes the successive maximal tick lengths, or reaction times, of a finite state synchronous program modelled by a TCA \( A \), then \( A[1] = \text{wcrt}(A) \) is the worst-case reaction time across all ticks. The value \( A[0] = A_0 \), which always exists, is the surface delay, i.e., the time from entering the program until the first pause is reached.

In the following, we let \( \mathbb{N}_\infty[X] \) denote the set of fps over \( \mathbb{N}_\infty \), which is max-plus algebra \( \mathbb{N}_\infty \) freely extended by a formal variable \( X \) to build denumerably infinite polynomials. It is important to keep in mind that the semantics of a fps \( F[X] \in \mathbb{N}_\infty[X] \) is not what it does as a function of \( X \), but the number series \( F_0, F_1, F_2, \ldots \) generated by it. In the following we use the fact that every fps \( A[X] \) can be uniquely written in the form \( A[X] = A_0 \oplus X A'[X] \) where the scalar \( A_0 \) is the initial coefficient (head) and \( A'[X] = A_1 \oplus A_2 X^1 \oplus A_3 X^2 \oplus \cdots \) is the first derivative (tail) of \( A[X] \) containing all the remaining coefficients.

### 3.2 Reducing TCAs to Linear Form

Now, if every TCA \( A \) corresponds to a formal power series \( A[X] \) then it should be possible to construct \( A \) from operators in \( \mathbb{N}_\infty[X] \). The idea is that each state \( q \in Q \) of \( A \) corresponds to a fps \( q[X] \) that describes the worst-case sequence of tick costs generated by \( A \) when started in \( q \). The fps for \( A \) then is \( A[X] = e[X] \) for the entry state \( e \) of \( A \). We need the following three operators, delay prefix, pause prefix and parallel composition on fps to represent every TAP\(^3\) as a system of recursive equations on fps:

- **Delay Prefix**
  
  The delay prefix composition \( A_0 \odot B[X] \) for a scalar \( A_0 \in \mathbb{N}_\infty \) and a fps \( B[X] = B_0 \oplus X B'[X] \) is given by
  
  \[
  A_0 \odot B[X] = (A_0 \odot B_0) \oplus X B'[X].
  \]

  The TCA \( A_0 \odot B[X] \) starts execution with a tick cost of \( A_0 \) and then instantaneously passes control to the TCA \( B \) for the rest of the current instant and all following ticks.

- **Pause Prefix**
  
  The pause prefix tick \( \text{tick}(A[X]) \) is defined by
  
  \[
  \text{tick}(A[X]) = 1 + X A[X].
  \]

  The TCA \( \text{tick}(A[X]) \) adds an initial pause node before starting \( A[X] \). It pauses the current tick and then behaves like \( A[X] \) from the second tick onwards.

- **Parallel Composition**
  
  The parallel composition \( A[X] \parallel B[X] \) of two fps \( A[X] = A_0 \oplus X A'[X] \) and \( B[X] = B_0 \oplus X B'[X] \) is given by
  
  \[
  (A_0 \oplus X A'[X]) \parallel (B_0 \oplus X B'[X]) = (A_0 \odot B_0) \oplus X (A'[X] \parallel B'[X]).
  \]

  The TCA \( A[X] \parallel B[X] \) executes the tick steps from \( A[X] \) and \( B[X] \) synchronously, but adds the tick costs to account for the interleaving at the level of the instantaneous transitions (multi-threaded semantics).

\(^3\)We recall that for simplicity the TCAs considered here do not have exit states. This permits us to describe TCAs and their parallel composition solely in terms of prefix operators.
Using these operators on $\mathbb{N}_\infty[X]$ together with the basic laws of max-plus algebra $\mathbb{N}_\infty$ we can construct for every TCA $A = \langle Q, e, \rightarrow \rangle$ an equivalent normal form representation $A^*$ of its corresponding fps $A[X]$. This normal form TCA $A^*$ is reduced since the only transient state is the entry state and it is linear since the transition relation has no branching.

We illustrate the formal transformations by way of the example automaton $A$ from Fig. 1. Considering each state as a generator fps $A_i = A_i[X]$ of tick costs and writing down the equations that arise from the transitions connecting them, we get:

\begin{align*}
A &= A_6 = (5 ; A_0) \oplus (8 ; \text{tick}(A_2)) \\
A_0 &= 7 ; \text{tick}(A_1) \\
A_1 &= 21 ; \text{tick}(A_2) \\
A_2 &= (31 ; \text{tick}(A_3)) \oplus (32 ; \text{tick}(A_4)) \\
A_3 &= (18 ; \text{tick}(A_3)) \oplus (20 ; \text{tick}(A_4)) \\
A_4 &= 24 ; A_5 \\
A_5 &= 12 ; \text{tick}(A_1)
\end{align*}

Let us explain the first equation (5), the others are constructed in a similar fashion. First note that the sum $\oplus$ in equation (5) expresses the non-deterministic choice between the two transitions out of state $A_6$, i.e., going to the transient node $A_0$ within 5 time units or to the surface $\text{tick}(A_2)$ of the pause state $A_2$ within 8 time units. In both cases, the operator $;$ prefixes the outgoing delay cost 5 or 8 to the target state $A_0$ or $\text{tick}(A_2)$, respectively.

Now note that the surface part $\text{tick}(A_2)$ of $A_2$ immediately returns control and then behaves like the depth part $A_2$ one tick later. This is expressed by the Pause Prefix Law (3) which gives $\text{tick}(A_2) = 1 \oplus X A_2$. Then, using the Delay Prefix Law (2) on the second transition, we get $A_0 = 8 ; \text{tick}(A_2) = 8 ; (1 \oplus X A_2) = (8 \circ 1) \oplus X A_2 = 8 \oplus X A_2$. Similarly, we get $A_0 = 7 ; \text{tick}(A_1) = 7 \oplus X A_1$ from the second equation (6). Substituting both into (5) yields $A_6 = (5 ; A_0) \oplus (8 ; \text{tick}(A_2)) = (5 ; (7 \oplus X A_1)) \oplus 8 \oplus X A_2 = (5 \circ 7) \oplus X A_1 \oplus 8 \oplus X A_2 = 12 \oplus X(A_1 \oplus A_2)$ with another application of Delay Prefix and the laws of max-plus algebra. This tells us that the worst case cost of the first tick of $A$ (started from $A_6$) is 12 and the remaining ticks are described by $A_1 \oplus A_2$. This makes sense since we do not know if we continue in $A_1$ or $A_2$, whence we must take the worst case, which is the maximum $\oplus$.

Continuing in this fashion, we obtain a normal form representation of $A[X]$, systematically evaluating and substituting the equations (5)–(11). Overall, we find:

$$A^*[X] = A_6[X] = 12 \oplus 32X \oplus 36X^2 \oplus 36X^3 \oplus \cdots$$

which corresponds to a reduced linear TCA $A^* \cong A$ as seen on the right of Fig. 1. From it we can read off the maximal tick length $\text{wcrt}(A) = \text{wcrt}(A^*) = A^*[1] = 36$.

As the above example illustrates, any TCA can be transformed into a reduced linear TCA using the laws of max-plus algebra and the (delay, pause) prefix laws. In general, a TCA in reduced linear form, also called a $l$–TCA, looks like

\begin{align*}
A &= A^* \oplus X^k A^\phi \\
A^* &= t_0 \oplus t_1 X \oplus \cdots \oplus t_k X^k \\
A^\phi &= r_0 X \oplus \cdots \oplus r_{n-1} X^n \oplus X^n A^\phi
\end{align*}
with an initial transient sequence $A^r$ and the recurrent iterative loop $A^\phi$. We call $\tau(A) = k$ the transient length and $\phi(A) = n$ the cycle length of $A$. Note, the transient length indicates the number of ticks needed before the TCA reaches its stationary cyclic behaviour. The special case of an l–TCA $A$ (12) in which $\tau(A) = 0$ is referred to as a monocyclic TCA, or m–TCA for short. The general reduced linear TCA is seen in Fig. 3.

For example consider the reduced linear TCA $A^*$ from Fig. 1. It has transient length $\tau(A^*) = 1$ and cycle length $\phi(A^*) = 1$. The transient part is $(A^*)^\tau = 12 \oplus 32$ and its recurrent part is $(A^*)^\phi = 36X \oplus X (A^*)^\phi$. The TCA $C$ from Fig. 2 is in reduced linear form with transient length of $\tau(C) = 2$ and cycle length $\phi(C) = 2$. The transient part is $C^\tau = 5 \oplus 1X \oplus 13X^2$ and the recurrent part $C^\phi = 2X \oplus 1X^2 \oplus X^2C^\phi$. Neither $A$ nor $C$ is monocyclic. On the other hand TCA $D$ in Fig. 2 is monocyclic.

**Proposition 3.1** Let $A$ be an arbitrary TCA, specified through a finite set of equations in $\mathbb{N}_\infty[X]$ with operations $\{\oplus, \odot, \tick\}$. Then, using the Delay and Pause Prefix Laws (2) and (3) together with the laws of max-plus algebra, $A$ can be transformed in polynomial time into a $l$–TCA $A^*$ of shape (12) with $A \sim A^*$.

Let us call a TAP (see Def. 2.4) in which all tick cost automata are $l$–TCAs an $l$–TAP. Similarly, a $m$–TAP is a TAP in which all automata are $m$–TCAs. Prop. 3.1 gives us a normalisation procedure to reduce an arbitrary TAP to an $l$–TAP of reduced linear TCAs, and then unfold until we are left with an $m$–TAP.

**Algorithm 3.2 (UNFOLD)** Given TCAs $T_i$ and a TAP $T = T_1\parallel T_2\parallel \cdots \parallel T_n$:

1. Use Prop. 3.1 to obtain the equivalent reduced linear form $T_i^*$ of each $T_i$.

2. Repeatedly use the Parallel Composition Law (4) on the TAP

   \[ T^* = T_1^* \parallel T_2^* \parallel \cdots \parallel T_n^* \]

   to factor out the transient parts of the $T_i^*$, i.e., to bring $T^*$ into the equivalent form $T^* = T^* \tau \oplus X^k T^* \phi$ where

   \[ T_i^* = T_1^* \parallel T_2^* \parallel \cdots \parallel T_n^* \]

   is an $m$–TAP in which all $T_i^* \phi$ are $m$–TCAs, i.e., such that $\tau(T_i^* \phi) = 0$. Each $T_i^* \phi$ is a cyclic shift of the recurrent part of the associated $T_i^*$, i.e., their cycle lengths are identical.

---

4Note that the term ‘transient’ here does not refer to instantaneous behaviour or transient states. It refers to the initial transient phase at the tick level.

5Every fps $F[X]$ has at least a cycle length of 1 since $F[X] = F[X] \oplus 0$ and $0 = 0 \oplus X^0$. 
UNFOLD reduces the problem of computing \( \text{wcrt}(T) \) for a TAP \( T = T_1 \parallel T_2 \parallel \cdots \parallel T_n \) involving arbitrary TCAs \( T_i \) to computing \( \text{wcrt}(T^{*,*}) \) for an \( l \)-TAP \( T^{*,*} \) (the transient part) and computing \( \text{wcrt}(T^{*,0}) \) for a TAP \( T^{*,0} = T_1^{*,0} \parallel T_2^{*,0} \parallel \cdots \parallel T_n^{*,0} \) consisting entirely of monocyclic TCAs \( T_i^{*,0} \). Both steps 1 and 2 of Alg. 3.2 are polynomial of asymptotic complexity \( \Theta(n \max(\tau^{*}_1, \ldots, \tau^{*}_n)) \) where \( \tau^{*}_i \) are the transient lengths of the \( T_i^{*,0} \) resulting from Step 1. The overall result is obtained by maximum,

\[
\text{wcrt}(T) = \max(\text{wcrt}(T^{*,*}), \text{wcrt}(T^{*,0})).
\]

The brute force approach to solve \( m \)-TAP instances is by reachability and state expansion.

**Algorithm 3.3 (EXPAND)** Given \( l \)-TCAs \( T_i \) and TAP \( T = T_1 \parallel T_2 \parallel \cdots \parallel T_n \). Repeatedly use the Parallel Expansion Law (4) to obtain an equivalent reduced linear form \( T^{*} \) of \( T \). Then,

\[
\text{wcrt}(T) = \text{wcrt}(T^{*}).
\]

Alg. 3.3 is of exponential complexity \( \Theta(n \lcm(\phi^{*}_1, \ldots, \phi^{*}_n)) \), where \( \phi^{*}_i \) are the cycle lengths of the \( T_i^{*,0} \).

![Figure 4: The parallel expansion (EXPAND) of \( E \parallel F \) (right) for the \( l \)-TCAs \( E \) and \( F \) (left).](image)

For illustration consider Fig. 4 which shows the result of applying EXPAND on the parallel composition \( E \parallel F \) of the \( l \)-TCAs seen on the left. Note how the cycle length of the composition is the product \( \phi(E \parallel F) = 6 = 2 \cdot 3 = \phi(E) \cdot \phi(F) \) while the transient length is the maximum \( \tau(E \parallel F) = 2 = \max(2, 0) = \max(\tau(E), \tau(F)) \). We can read off the WCRT as \( \text{wcrt}(E \parallel F) = 5 \odot 14 = 19 \).

For \( m \)-TAP instances there is a standard case in which a parallel operator can be eliminated in polynomial time, viz. if the cycle length of one \( m \)-TCA divides that of another.

**Proposition 3.4** Let \( T_i = \bigoplus_{j=0}^{\phi_i-1} t_{ij} X^{j+1} \oplus X^{\phi_i} T_i \) for \( i = 1, 2 \) be two \( m \)-TCAs such that \( \phi_1 \) divides \( \phi_2 \), i.e., \( \gcd(\phi_1, \phi_2) = \phi_1 \). Then, \( T_1 \parallel T_2 \cong T \) where \( \tau(T) = 0, \phi(T) = \phi_2 \) and

\[
T = \bigoplus_{j=0}^{\phi_2-1} (t_{1(j \bmod \phi_1)} + t_{2j}) X^{j+1} \oplus X^{\phi_2} T.
\]
The equivalence reduction $T_1 \parallel T_2 \cong T$ of Prop. 3.4 can be conducted in PTIME, e.g., by repeated application of the Parallel Expansion Law. This gives rise to the REDUCE algorithm which can be used in Alg. 3.2 as part of Step 2 to simplify the recurrent part $T^{\ast \phi}$ further.

**Algorithm 3.5 (REDUCE)**

```latex
f = 0
i = 1
while i < len(T) :
  if len(T[i]) < len(T[f]) :
    T[i], T[f] = T[f], T[i]
  if len(T[i]) % len(T[f]) == 0:
    T[f] = x + y for x, y in zip(T[f] *(len(T[i])/len(T[f])), T[i])
  T = T[-1]. T[i]
  i = i + 1
return T
```

The algorithm 3.5 can be thought of as being composed of two premiere actions: duplication and fusion. If the cycle length of one m–TAP divides the length of another, then the shorter one is multiplied by duplicating transitions to have the same number of transitions as the longer one, and these two m–TCAs are then fused into a single m–TCA by summing their transition costs.

Where such an m–TAP reduction is not possible we are facing the complexity problem of the Parallel Expansion in Step 3 of Alg. 3.3 which bears the risk of a state-space explosion. So, for efficiency, we cannot eliminate parallel composition completely in Step 3 but instead need other techniques which preserve some degree of concurrency. To this end, it will be helpful to transform m–TAPs into an equivalent graph-theoretic maximum cost clique problem based on the number-theoretic structure of the associated m–TCAs.

4 The Tick Alignment Graph and Maximum Weight Cliques

From now on we generally assume that $T = T_1 \parallel T_2 \cdots \parallel T_n$ is an m–TAP, i.e., all TCAs $T_i$ are monocyclic. Each m–TCA $T_i$ can be identified with a function that associates a tick cost $T_i(j) \in \mathbb{N}$ with each index $0 \leq j < \phi_i$ representing the $j$-th transition counted from the start of the m–TCA, called a transition offset. For instance, for the m–TCA $A^\phi$ in Fig. 3 we have $A^\phi(j) = r_j$. Then, the problem of computing $\text{wcrT}(T)$ amounts to finding the the maximum sum $T_1(t_1) + T_2(t_2) + \cdots + T_n(t_n)$ for any selection of transition offsets $0 \leq t_i < \phi_i$ that are tick aligned, i.e., for which there exists a global tick count $k$ such that $k \equiv \phi_i \mod m$. Formally, then

$$\text{wcrT}(T) = \max\{T_1(t_1) + T_2(t_2) + \cdots + T_n(t_n) \mid \exists k \geq 0. \forall 1 \leq i \leq n. 0 \leq t_i < \phi_i, k \equiv \phi_i \mod m\}.$$

Consider the m–TAP $T = T_1 \parallel T_2 \parallel T_3 \parallel T_4$ seen in Fig. 5, where the m–TCA $T_1$ has the cycle length $\phi(T_1) = 3$ and in max-plus notation is $T_1 = 1X + 4X^2 + 5X^3 + X^3T_1$ or as a tick cost function $T_1(0) = 1, T_1(1) = 4$ and $T_1(2) = 5$. The other three m–TCAs $T_2, T_3,$ and $T_4$ are specified in an analogous way. If we start each TCA $T_i$ in its entry state, indicated by the small horizontal arrow, then in the $k$-th tick it executes the transition with cost $T_i(k \mod \phi(T_i))$. Consequently, as highlighted by the red dotted line in Fig. 5 the sum $T_1(2) + T_2(0) + T_3(2) + T_4(0) = 5 + 2 + 1 + 3 = 11$ is aligned since for $k = 2$
the TCAs $T_1$ and $T_3$ execute the transition with offset $k \mod 3 = 2$ and TCAs $T_2$ and $T_4$ both execute the transition with offset $k \mod 2 = 0$. On the other hand, the sum $\text{wcrt}(T_1) + \text{wcrt}(T_2) + \text{wcrt}(T_3) + \text{wcrt}(T_4) = 5 + 2 + 3 + 3 = T_1(2) + T_2(0) + T_3(0) + T_4(0) = 13$ is not aligned: There is no global tick count $k$ such that $k \mod 3 = 2$ and at the same time $k \mod 3 = 0$ which would be necessary to make $T_1$ and $T_3$ to reach their locally maximal tick costs $T_1(2) = 5$ and $T_3(0) = 3$ simultaneously.

Figure 5: An m–TAP instance composed of 4 threads. Its WCRT is 9 and occurs when the transitions $T_1(2)$, $T_2(0)$, $T_3(2)$ and $T_4(0)$ are executed.

**Proposition 4.1 (Chinese Remainder Theorem)**

Given an m–TAP $T = T_1 || T_2 || \cdots || T_n$ where $\phi_i = \phi(T_i)$ is the cycle length of $T_i$ for $1 \leq i \leq n$. A candidate sum

$$T_1(t_1) + T_2(t_2) + \cdots + T_n(t_n),$$

with transition offsets $0 \leq t_i < \phi_i$, is aligned in $T$ iff for all pairs of indices $1 \leq i_1, i_2 \leq n$, we have $t_{i_1} \equiv \text{gcd}(\phi_{i_1}, \phi_{i_2}) t_{i_2}$.

Proposition 4.1 suggest a decision procedure. We build a tick alignment graph connecting a transition $t_{i_1}$ of one m–TCA with a transition $t_{i_2}$ from another m–TCA iff $t_{i_1} \mod g_{(i_1,i_2)} = t_{i_2} \mod g_{(i_1,i_2)}$, where $g_{(i_1,i_2)} = \text{gcd}(\phi(T_{i_1}), \phi(T_{i_2}))$. We then search for a fully connected subset of transitions, one from each m–TCA with maximal weight. Since every transition in each thread is connected (aligned) with some transition in every other thread, this is the same as searching for a maximal weight clique.

Let us make this more precise. Let $G = (V, E, w)$ be a finite undirected graph with vertices $V$, symmetric and reflexive edge relation $E \subseteq V \times V$ and node weights $w : V \to \mathbb{N}$. We write $v_1 \leftrightarrow_E v_2$ for $(v_1, v_2) \in E$. A subset $C \subseteq V$ is a clique if it is fully connected, i.e., for all $v_1, v_2 \in C$, $v_1 \leftrightarrow_E v_2$. The weight $w(S)$ of a subset $S \subseteq V$ is $w(S) \overset{df}= \sum\{w(v) \mid v \in S\}$.

**Definition 4.2 (Tick Alignment Graph – TAG)**

Let $T = T_1 || T_2 || \cdots || T_n$ be an m–TAP with cycle lengths $\phi_i = \phi(T_i)$ and pairwise greatest common divisors $g_{(i_1,i_2)} \overset{df}= \text{gcd}(\phi_{i_1}, \phi_{i_2})$ for $1 \leq i, i_1, i_2 \leq n$. The tick alignment graph (TAG) $G_T = (V_T, E_T, w_T)$ induced by $T$ is defined by:

- $V_T \overset{df}= \{(i, j) \mid 1 \leq i \leq n, 0 \leq j < \phi_i\}$
- $(i_1, j_1) \leftrightarrow_{E_T} (i_2, j_2)$ iff $j_1 \equiv g_{(i_1,i_2)} j_2$
- $w_T(i, j) \overset{df}= T_i(j)$.
For our example m–TAP $T$ from Fig. 5 the tick alignment graph $G_T$ is shown in Fig. 6 on the top. The nodes of $G_T$ representing transitions of $T$ are drawn as boxes to distinguish them from the states of the m–TCAs in Fig. 5. Inside each box we have written the formal representation $(i,j)$ of the vertex. Its weight $w_T(i,j)$ is given by the number above the box. The edges in the TAG connect nodes $(i_1,j_1)$ and $(i_2,j_2)$ iff the thread offsets satisfy $j_1 \equiv g_{(i_1,i_2)} \mod j_2$ where $g_{(i_1,i_2)} = \gcd(\phi_{i_1}, \phi_{i_2})$ and $\phi_i = \phi(T_i)$. These “connectivity parameters” are given in the graph seen on the bottom of Fig. 6.

Notice in Fig. 5 that no two nodes of the same thread are connected. Indeed, they can never occur together in the same tick. Formally, this is because $j_1 \neq \phi_j$ for all $j_1 \neq j_2$, considering that $\phi_i = g_{(i,i)} = \gcd(\phi_i, \phi_i)$. As a consequence we have $(i, j_1) \not\in E_T$ $(i, j_2)$ for all $1 \leq i \leq n$ whenever $j_1 \neq j_2$. On the other hand, as can be seen, threads with relatively prime cycle lengths have all their nodes fully connected between them. For instance, $g_{(1,2)} = \gcd(\phi_1, \phi_2) = \gcd(3,2) = 1$ so that $j_1 \equiv g_{(1,2)} \equiv g_{(1,2)} \mod j_2$ for any $j_1$ and $j_2$. So, e. g., all nodes $(1,j_1)$ in $T_1$ are connected with all nodes $(2,j_2)$ in $T_2$.

Given these connections in $G_T$, a clique $C = \{(1,2), (2,0), (3,2), (4,0)\}$ with maximal weight $W_T(C) = 11$ is highlighted in Fig. 6 by thick red lines. Another clique with weight 10 is $C' = \{(1,1), (2,0), (3,1), (4,0)\}$. These cliques correspond to the two aligned candidate sums $T_1(2) + T_2(0) + T_3(2) + T_4(0)$ and $T_1(1) + T_2(0) + T_3(1) + T_4(0)$ and the two ways of generating the tick cost of the composite TCA $T_1\|T_2\|T_3\|T_4$ for tick counts $k = 2$ and $k = 4$, respectively.

![Figure 6: The tick alignment graph for the TAP from Fig. 5 and a clique of maximal weight 11 on the top. The graph on the bottom indicates the cycle lengths $\phi_i$ of the four threads $T_i$ and the greatest common divisors $g_{i,j} = \gcd(\phi_i, \phi_j)$ connecting them.](image)

**Proposition 4.3 (Reduction to Max Weight Clique Problem (MWCP))**

Let $T = T_1\|T_2\|\cdots\|T_n$ be an m–TAP with cycle lengths $\phi_i = \phi(T_i)$ and $m = \max_i \phi_i$ their maximum.

- The associated TAG $G_T$ can be computed in $O(n^2m^2)$ time and has size $O(nm)$ vertices and $O(n^2m^2)$ edges.
• A candidate sum of the m–TAP $T$,

$$T_1(t_1) + T_2(t_2) + \cdots + T_n(t_n),$$

for $0 \leq t_i < \phi_i$, is aligned in $T$ iff the nodes $C = \{(i, t_i) \mid 1 \leq i \leq n\}$ form a clique in the TAG $G_T$.

• To check a candidate sum $C$ is a clique takes $O(n^2)$ time.

• $wcr(T) = \max\{w_T(C) \mid C \text{ clique in } G_T\}$.

Prop. 4.3 reduces the TAP for m–TAP instances to the Maximum Weight Clique Problem (MWCP), which is known to be NP-complete for arbitrary graphs [12]. This means that TAP is in NP which is already more information than we get from Alg. 3.3 which only shows that TAP is in EXPTIME. However, this is still unsatisfactory because it does not provide a lower bound on the computational complexity of TAP.

In the appendix (Sec. A) we show that it is possible to reduce any instance of MWCP for an arbitrary graph $G$ with $n$ nodes to a m–TAP $T_G$. However, this recoding depends on the generation of $O(n^2)$ distinct prime numbers $p_i$, obtaining an instance $T_G$ of TAP of size $O(n \Pi p_i)$ if the cycles are represented explicitly. This is an exponential space blow-up in the m–TAP instance and so the reduction does not imply NP-hardness of m–TAP. Also, even if the transitions of the TAP $T_G$ were coded implicitly in polynomial space, it is not clear if the generation of $n$ distinct prime number is in PTIME. Certainly the generation of the $n$ first distinct prime numbers is highly unlikely to be PTIME for otherwise the number factorisation problem would be in PTIME, too, which is believed not to be the case (RSA encryption would be pointless if it were).

This suggests that m–TAP—or MWCP on tick alignment graphs, for that matter—may well be polynomial in practice. After all, the tick alignment graphs generated from TAP instances are not arbitrary graphs but have specific structure arising from the number-theoretic relationships of the cycle lengths involved. One important such special property, which we will exploit later, is captured by the following Prop. 4.4.

**Proposition 4.4** Let $T = T_1 \parallel T_2 \parallel \cdots \parallel T_n$ be an m–TAP with cycle lengths $\phi_1 = \phi(T_i)$. Every clique $C \subseteq V_T$ of $G_T = (V_T, E_T, w_T)$ can be extended to a clique $C' \supseteq C$ containing one node from each thread, i.e., for all $1 \leq i \leq n$ there is a $0 \leq j < \phi_i$ with $(i, j) \in C'$.

## 5 Practical Algorithms for TAP

As an application of the theoretical results from the previous sections we now describe our experiments with practical algorithms both for the exact solution of the TAP, in Sec. 5.1, as well as polynomial approximations, in Sec. 5.2. We start with some terminology regarding the proposed algorithms in this section.

• Exact algorithms: These algorithms are optimal but exhibit an exponential worst-case behaviour. We start with an algorithm for the MWCP called $wclique$. Following this we present an improved algorithm called ILP$_{CP}$ which extends an iterative narrowing based algorithm called ILP$_C$ using pairwise constraints during the narrowing process that converges faster.
Approximation algorithms: These algorithms are PTIME at the expense of a loss of precision. We propose an approach called MaxCy to compute the maximum cost cycle in a TAG. We also propose a variant of MaxCy called MaxCy+Reduce that applies the technique presented in Algorithm 3.5. Finally, we compare our techniques to existing techniques based on forming the sum of the maximum tick costs in the threads, called MaxTC.

5.1 Exact Computation of WCRT

Many algorithms have been proposed to solve the MWCP. The most well-known are encodings in Integer Linear Programming (ILP) style, see e.g. [12], or branch-and-bound search algorithms such as [11, 17]. All these can be applied to obtain exact solutions for the TAP via Prop. 4.3. Though these algorithms, solving an NP-complete problem, have exponential worst-case behaviour on arbitrary graphs, it is not known how they fare on tick alignment graphs. We conducted experiments to find out and the results are reported in Sec. 6.

Alongside, we observe that the incremental ILP algorithm [15], an effective WCRT evaluation method, is also based on similar approaches. Starting from this observation, we compare the linear programming formulation of ILP with the most common linear formulation of the MWCP. This results in a simple but extremely efficient improvement of ILP, which we term as ILP_{CP}.

5.1.1 The Global ILP Formulation of MWCP

For a weighted graph $G = (V, E, w)$, let $\overline{G} = (V, \overline{E}, w)$ be the complement graph such that $\overline{E} = \{(u, v) \mid u, v \in V, u \neq v \text{ and } (u, v) \notin E\}$. Then, if $G_T = (V_T, E_T, w_T)$ is the TAG induced by a TAP $T$, according to Prop. 4.3, we obtain an exact ILP solution for wcrt($T$) from the following zero-one linear program, which is probably one of the most common formulation of the MWCP on $G_T$, called the edge formulation [12]:

**Algorithm 5.1 (ILP_{MWCP})**

Maximize $\sum_{(i,j) \in V_T} w_T(i,j) \cdot E_i(j)$ with \[
E_i(j_1) + E_i(j_2) \leq 1, \text{ for all } (i_1, j_1) \leftrightarrow_{E_T} (i_2, j_2)\\E_i(j) \in \{0, 1\}, \text{ for all } (i, j) \in V_T.
\]

Recall that for a thread offset $(i, j) \in V_T$, i.e., $0 \leq j < \phi_i$, the value $w_T(i, j) = T_i(j)$ is its tick cost. Each $E_i(j)$ is a Boolean variable that indicates if the offset is part of the selected combination. So, each constraint $(i_1, j_1) \leftrightarrow_{E_T} (i_2, j_2)$, by modelling the absence of a connection in the TAG $G_T$, ensures that the two corresponding thread tick costs $T_i(j_1)$ and $T_{i_2}(j_2)$ will not be considered as aligned, i.e., active during the same tick. All thread offsets selected by the $E_i(j)$ satisfying the constraints form a clique.

5.1.2 The Iterative ILP{C} Approach

The iterative ILP{C} algorithm [15] starts from a relaxation of the above linear program, called ILP_{BASE}:

**Algorithm 5.2 (ILP_{BASE} or Maximum Thread Cost Approach)**

Maximize $\sum_{(i,j) \in V_T} w_T(i,j) \cdot E_i(j)$ with \[
\sum_{j=0}^{\phi(T_i)-1} E_i(j) = 1, \text{ for all } i \in \{1, \ldots, n\}\\E_i(j) \in \{0, 1\}, \text{ for all } (i, j) \in V_T.
\]
This linear program ILP_{BASE} is equivalent to solving the above ILP_{MWCP} on the relaxed TAG $G_T^*$ in which all offsets between different threads are connected. This only ensures that transitions in the same thread are not considered together for a candidate sum. Since $G_T^*$ is an edge extension of $G_T$, the result of ILP_{BASE} is an upper bound over-approximation of the result from ILP_{MWCP}. In fact, ILP_{BASE} can be computed in PTIME and gives the same WCRT as the Maximum Thread Cost (MaxTC) approach [4, 10].

The maximum weight solution $\sum_{(i,j) \in V_T} w_T(i,j) \cdot E_i(j)$ obtained from ILP_{BASE} as a candidate sum may or may not be aligned. More precisely, let $C = \{(i,j) \mid E_i(j) = 1, (i,j) \in V_T\}$ be the candidate set selected in ILP_{BASE}. Then, $\text{wcr}(T) = w_T(C)$ iff $C$ is a clique in $G_T$ which can be checked in PTIME, see Prop. 4.3. If $C$ is not a clique, then the strategy of ILP_{C} is to improve the ILP_{BASE} model iteratively by adding new constraints that rule out these detected infeasible combinations until finally reaching a valid one. The infeasibility of tick combinations $C$ is confirmed using another linear program called ILP_{CHECK}.\(^{6}\)

For example, on the previous TAP of Fig. 5, solving the ILP_{BASE} problem might result in the candidate set $C = \{(1,2), (2,0), (3,0), (4,0)\}$ selecting from each thread an offset of maximum cost, thereby yielding the total tick costs $T_1(2) + T_2(0) + T_3(0) + T_4(0) = 13$. Using ILP_{CHECK}, we detect it is an infeasible combination. Specifically, the nodes $(1,2)$ and $(3,0)$ are not aligned in $G_T$ because $2 \neq 3 \mod 3$ where $3 = \gcd(3,3) = \gcd(\phi(T_1), \phi(T_3))$. Therefore, the following constraint (13) is added to the ILP_{BASE} problem:

$$E_1(2) + E_2(0) + E_3(0) + E_4(0) < 4.$$  \hspace{1cm} (13)

This constraint expresses that these offsets will never be activated together at the same time. By solving ILP_{BASE} a second time, the candidate set $C$ will be excluded and the maximal solution is given by the candidate set $C' = \{(1,2), (2,0), (3,0), (3,1)\}$ with weight $w_T(C') = 13$. Again, ILP_{CHECK} finds out this set is infeasible and generates the constraint

$$E_1(2) + E_2(0) + E_3(0) + E_4(1) < 4.$$  \hspace{1cm} (14)

Now, when ILP_{BASE} is iterated for the third time under both (13) and (14), in this case, the exact solution appears.

5.1.3 Improved Iterative ILP_{CP} Method

By taking into account our theoretical analysis, we improve the ILP_{C} method in two points:

- We propose a polynomial alternative to ILP_{CHECK}.
- We strengthen the infeasible combination constraint using the MWCP formulation.

In the proposed polynomial version of ILP_{CHECK}, we check all pairs of offsets for alignment, i.e., for a connection in the TAG. Each missing edge in $G_T$ not only witnesses the infeasibility of the given candidate sum but also of others. We can take advantage of this information to tighten up the ILP_{BASE} so we need fewer iterations. Note that while ILP_{C} only ruled out a specific candidate sum, we propose to rule out every candidate sum that shares some infeasible pair of offsets with the specific candidate sum currently under check. These new pairwise constraints are then exactly corresponding to those of the edge formulation of MWCP.

\(^{6}\)The programs ILP_{BASE} and ILP_{CHECK} generate and check, respectively, tick alignment not just for tick costs in m-TAPs but general sequential-parallel program structures.
If \( C \) is any combination of nodes in a TAG \( G_T \), we denote by \( \Xi(C) \) the list of infeasible pairs of thread offsets from \( C \). Formally, we define for each \( C \subseteq V_T \)
\[
\Xi(C) \overset{df}{=} \{(i_1,j_1),(i_2,j_2)\mid (i_1,j_1),(i_2,j_2) \in C \text{ and } (i_1,j_1) \leftrightarrow_{E_T} (i_2,j_2)\}.
\]
From these we generate the ILP constraints
\[
E_{i_1}(j_1) + E_{i_2}(j_2) \leq 1, \text{ for all } ((i_1,j_1),(i_2,j_2)) \in \Xi(C)
\] (15)
to narrow the \( \text{ILP}_{BASE} \) formulation for better precision. Note that if we choose \( C = V_T \) then we have maximum precision, since we are completing \( \text{ILP}_{BASE} \) to become equivalent to \( \text{ILP}_{MIVCP} \). The fact that \( \Xi(C) \) is obtained from a small candidate set \( C \) which generates an upper bound on the WCRT make this approach more focused than Algorithm 5.1.

For example, in Fig. 5, considering the infeasible candidate sum \( T_1(2) + T_2(0) + T_3(0) + T_4(0) \), we have \( \Xi(\{(1,2),(2,0),(3,0),(4,0)\}) = \{(1,2),(3,0)\} \). Now just have to add the constraint \( E_1(2) + E_3(0) \leq 1 \) to \( \text{ILP}_{BASE} \). This constraint is not only valid for all cliques, it is also stronger than the constraint (13) used in \( \text{ILP}_{CHECK} \). That is, it rules out the infeasible sum \( T_1(2) + T_2(0) + T_3(0) + T_4(0) \) like (13) does, and also \( T_1(2) + T_2(0) + T_3(0) + T_4(1) \) like (14), as well as \( T_1(2) + T_2(1) + T_3(0) + T_4(0) \) and \( T_1(2) + T_2(1) + T_3(0) + T_4(1) \). In this way only one iteration is needed to hit the exact solution.

5.2 MaxCY Polynomial Approximation

Producing the exact solution is not always a requirement. Good over-estimations may be tight enough. One canonical way to obtain approximations of wcrt\((T)\) for a \( m \)-TAP instance \( T \) proceeds by computing the maximum weight of all \textit{quasi-cliques} in the associated TAG \( G_T \), where a quasi-clique is some suitable relaxation of the clique structure as defined in the following Prop. 5.3.

Proposition 5.3 Let \( \text{QuasiClique} \subseteq 2^{V_T} \) be a class of subsets of nodes of a TAG \( G_T = (V_T,E_T,w_T) \) which contains all cliques of \( G_T \) in the following sense: For every clique \( C \subseteq V_T \) there exists an extension \( C' \supseteq C \) with \( C' \in \text{QuasiClique} \). Then, \( \text{wcrt}(T) \leq \max\{w_T(C) \mid C \in \text{QuasiClique} \} \).

A notion of quasi-clique in Prop. 5.3 is interesting for WCRT analysis if the maximum weight quasi-clique can be found in \( \text{PTIME} \). Two simple examples are the following:

- **(MaxAll)** Considering every subset \( C \subseteq V_T \) as a quasi-clique, then the maximum weight quasi-clique is \( V_T \) which simply yields the sum of all costs \( \sum_{v \in V_T} w_T(v) = \sum_{i,j} T_i(j) \) which is a trivial upper bound of wcrt\((T)\).

- **(MaxTC)** A tighter result is achieved by taking a quasi-clique to be a subset \( C \subseteq V_T \) that contains at most one node from each thread. Formally, \( C \in \text{QuasiClique} \) iff \((i,j_1) \in C \) and \((i,j_2) \in C \) implies \( j_1 = j_2 \). The maximum weight such quasi-clique \( C_{max} \) then arises from the selection of the maximum offset from each thread. In other words, \( \max\{w_T(C) \mid C \in \text{QuasiClique} \} = \sum_i \max_j T_i(j) \) which is nothing but the Maximum Thread Cost approach [4, 10], also specified by \( \text{ILP}_{BASE} \).

There are even more constrained notions of quasi-cliques that can be maximised in \( \text{PTIME} \) such as \textit{bi-partite} sub-graphs or \textit{triangulated} sub-graphs. For a discussion and review of literature see [12]. Here we propose a new approach, called MaxCY, based on \textit{(directed) cycles} as a notion of quasi-clique. To this end, we fix an arbitrary total ordering, or permutation,
\[
\pi = [T_{\pi(1)}, T_{\pi(2)}, \ldots, T_{\pi(n)}]
\]
on the threads of an \( m \)-TAP instance \( T = T_1 \| T_2 \| \cdots \| T_n \). This generates a cyclic structure on the threads which can be used to direct the edges in the tick alignment graph \( G_T \).
Definition 5.4 (Directed Neighbourhood TAG) Let \( G_T \overset{df}{=} (V_T, E_T, w_T) \) be a TAG and \( \pi \) a permutation on \( \{1, 2, \ldots, n\} \). We define the \( \pi \)-directed neighbourhood TAG \( G_T^{\pi} \overset{df}{=} (V_T, E_T^{\pi}, w_T) \), with the same nodes and weights as \( G_T \), by selecting the edge directions \( E_T^{\pi} \subseteq E_T \) according to \( \pi \), i.e., \((i_1, j_1) \rightarrow E_T^{\pi} (i_2, j_2) \) iff \((i_1, j_1) \leftrightarrow E_T (i_2, j_2) \) and there exists a \( 1 \leq i \leq n \) such that \( i_1 = \pi(i) \) and \( i_2 = \pi(i + 1 \ mod \ n) \).

Def. 5.4 creates a directed neighbourhood version \( G_T^{\pi} \) of \( G_T \) in which we can search for cycles. Note that in the original graph the edge set \( E_T \) is symmetric by definition. What \( E_T^{\pi} \) then is doing is to select those directions from the symmetric pairs that are compatible with the sequence \( \pi(1), \pi(2), \ldots, \pi(n) \) induced by the permutation.

Let us call a cycle \( C \subseteq V_T \) in \( G_T^{\pi} \) a \( \pi \)-cycle of \( G_T \). One can show that \( \pi \)-cycles form a notion of quasi-cliques according to Prop. 5.3. This follows since every clique in \( G_T \) can be extended to a clique \( C' \supseteq C \) which contains one node from each thread, by Prop. 4.4. This clique \( C' \) then is a \( \pi \)-cycle, for any permutation \( \pi \), because it is fully connected.

The computation of the maximal weight \( \pi \)-cycle is very similar to the Maximum Cycle Mean (MCM) Problem for which there exist several PTIME algorithms. A comparative study of existing algorithms is available in [6]. The soundness of Alg. 5.5 follows from Prop. 5.3.

Algorithm 5.5 (MaxCY) Let \( T \) be an \( m \text{-}TAP \) instance. Select a total thread ordering \( \pi \) and transform \( T \) into its \( \pi \)-directed neighbourhood TAG \( G_T^{\pi} \). Then, \( \text{wcrt}(T) \leq \max \{ w_T(C) \mid C \text{ cycle in } G_T^{\pi} \} \).

Consider the TAG \( G_T \) from Fig. 6 with the natural ordering \( \pi_0 = [T_1, T_2, T_3, T_4] \). The resulting \( \pi_0 \)-directed TAG \( G_T^{\pi_0} \) is shown in Fig. 7. Also, by thick red lines, a cycle \( C = \{(1, 2), (2, 0), (3, 0), (4, 0)\} \) with maximal weight \( w(C) = 13 \) is outlined. TWe know this is an over-estimation of \( \text{wcrt}(T) \) as the optimal tick cost is 11. It is not a clique because nodes (1, 2) and (3, 0) are not connected in \( G_T \), for instance. If we had chosen a different ordering such as \( \pi_1 = [T_1, T_3, T_2, T_4] \) then the set \( C = \{(1, 2), (2, 0), (3, 2), (4, 0)\} \) of weight 10 would not be a cycle. Instead, the directed neighbourhood graph \( G_T^{\pi_1} \) has the maximal weight cycle \( C' = \{(1, 2), (3, 2), (2, 0), (4, 0)\} \) of weight \( w(C') = 11 \) which is exact.

![Figure 7](image)

Figure 7: The \( \pi_0 \)-directed neighbourhood TAG \( G_T^{\pi_0} \) for \( G_T \) of Fig. 6 together with a cycle of maximal weight 13 (left) and the \( \pi_1 \)-directed neighbourhood TAG \( G_T^{\pi_1} \) with a cycle of maximum weight 11 (right).

Finally, we propose an improvement over MaxCY called MaxCY+Reduce. This reduces the sensitivity of Alg. 5.5 to the ordering. Here, the \( m \text{-}TAP \) instance is simplified using Alg. 3.5 before the directed neighbourhood TAG is created. This simple transformation has no effect on the overall complexity of a \( m \text{-}TAP \) instance (as the greatest common
divisor of thread lengths is the same), but we experimentally find that it significantly reduces the order effect. The reason is that by merging threads of commensurable length we ensure that they will be considered as direct neighbours in the search for cycles.

This can be seen clearly in our example TAG of Fig. 6 for $T = T_1\parallel T_2\parallel T_3\mid T_4$ in which the pair of threads $T_1$ and $T_3$ as well as the pair $T_2$ and $T_4$ each have the same cycle length. Each of these pairs is merged into a single m–TCA by Alg. 3.5, say $T_1\parallel T_3 \cong T_{13}$ and $T_2\parallel T_4 \cong T_{24}$, as seen in Fig. 8. As a result, in the reduced m–TCA $T^* = T_{13}\parallel T_{24}$ the indicated clique of maximum weight 11 is also a maximum weight cycle in $G^*_T$ for any ordering $\pi$.

![Figure 8: The m–TAP instance of Fig. 6 reduced by Alg. 3.5 as $T = T_1\parallel T_2\parallel T_3\mid T_4 \cong T_{13}\parallel T_{24} = T^*$.](image)

### 6 Results

We start our evaluation of the proposed and existing optimal/exact techniques. Hence, we evaluate the performance and precision of solving TAP as a Maximal Weighted Clique Problem [17, 11] as described in Section 4 and compare this method with ILP$_C$ [15] and StateExploration [9, 18]. ILP$_{CP}$, our improved version of ILP$_C$, is also considered.

In a second part, we compare the approximate methods proposed in Sec. 5 with the only other existing approximate method, the Maximum Thread Cost [4, 10].

These experiments was did both on real-life and synthetics benchmarks. The real-life benchmark come from a set of synchronous applications previously proposed by Wang et al. [15] from which it was easy to extract TAP instances (names and sizes of these applications is presented in Table 1). On the other hand, we also produced a synthetic benchmark of exactly 8000 m–TAP of varying complexity, i.e., the least common multiple of the cycle lengths of the constituent TCAs (as discussed in Sec. 3) with a range of thread numbers, of thread sizes and of state duration between 1 and 20, all randomly determine using an uniform distribution. This benchmark allow use to evaluate both scalability and accuracy of our methods. It was not needed to produce samples of bigger size, as the complexity of TAP problem doesn’t come from the size of an application but from the overall lcm of its thread sizes.

#### 6.1 WClique and ILP$_{CP}$: Two exact methods

The Maximal Weighted Clique Problem was extensively studied in the past, and there already exist several effective algorithms to solve it. The WClique program [11] is one of them and publicly available.
In order to obtain an accurate performance estimation of our proposed method, we used our synthetic benchmarks. We then compare the computation time of WClique with the ILP$_C$ algorithm proposed by Wang et al. [15] (see Sec. 5.1.2), a full state exploration and ILP$_CP$, our proposed improvement of ILP$_C$ (see Sec. 5.1.3).

The Fig. 9 presents the results from our evaluation. Every point is corresponding to a particular instance of TAP and the lines show the average trend of each evaluated method. It is interesting to note that, as the complexity of the instances is growing, the performance of WClique are revealing better than the state exploration. Furthermore, WClique is also superior to ILP$_C$ in any case. This can be explained from the fact that ILP$_C$ is a domain-specific algorithm designed to fully solve general instances of TAP not just m–TAP. On the other hand, WClique aims to solve MWCP, which is a different, possibly more general problem. We are yet to show the exact complexity equivalence of MWCP and TAP.

6.2 MaxCY and Polynomial Approximations

As the directed neighbourhood graph method MaxCY (see Sec. 5.2) is an approximation algorithm, we do not expect optimal results. Therefore, it is crucial to evaluate its precision in relative terms, comparing with other polynomial algorithms, and in absolute terms how far off it falls from the exact WCRT. For this purpose, we present two different kinds of evaluations, the first based on TAP instances derived from real-life synchronous applications and the second using synthetically generated instances of m–TAP.

For the first set of experiments, we selected the TAP instances extracted from the applications proposed by Wang et al. [15]. We compared the proposed polynomial method
MaxCY in combination with UNFOLD (Alg. 3.2) relative to the Maximum Thread Cost (MaxTC) method \cite{4,10} and exact solutions computed using any exact method as ILP\textsubscript{CP}.

<table>
<thead>
<tr>
<th>Name</th>
<th>#threads</th>
<th>MaxTC</th>
<th>MaxCY+U</th>
<th>Exact</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChannelProtocol</td>
<td>7</td>
<td>21</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>Flasher</td>
<td>13</td>
<td>39</td>
<td>37</td>
<td>37</td>
</tr>
<tr>
<td>RobotSonar</td>
<td>7</td>
<td>35</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>Synthetic1</td>
<td>7</td>
<td>35</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>Synthetic2</td>
<td>7</td>
<td>35</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>DrillStation</td>
<td>19</td>
<td>226</td>
<td>170</td>
<td>170</td>
</tr>
<tr>
<td>CruiseControl</td>
<td>36</td>
<td>108</td>
<td>91</td>
<td>91</td>
</tr>
<tr>
<td>RailroadCrossing</td>
<td>46</td>
<td>276</td>
<td>205</td>
<td>205</td>
</tr>
<tr>
<td>WaterMonitor</td>
<td>45</td>
<td>225</td>
<td>126</td>
<td>126</td>
</tr>
<tr>
<td>Overestimate</td>
<td></td>
<td>34%</td>
<td>0%</td>
<td>0%</td>
</tr>
</tbody>
</table>

Table 1: Evaluation of our approximation on real-life applications. MaxCY+U includes UNFOLD and MaxCY.

Comparing to the only existing approximate method MaxTC by giving the exact solution at each time, our method seems to be extremely efficient, on these particular benchmarks. If this is a general pattern this would indicate that the state space explosion problem does not happen for real-life synchronous programs. Note that in these benchmarks the TCAs are not monocyclic and therefore MaxCY+U includes UNFOLD to separate the transient from the recurrent part of the TAP and taking the maximum over both parts.

Our second set of experiments, based on the synthetic benchmarks, attempted a more exhaustive analysis of the precision of the MaxCY approximation method without the bias to synchronous programs. Indeed, as we have seen in Sec. 5.2, MaxCY is rather sensitive to the thread ordering used in the cycle search. To alleviate this we introduce the REDUCE method (Alg. 3.5) to increase the coupling in the directed neighbourhood graph.
Fig. 10 presents the results of these experiments. As is clearly observable, on the synthetic benchmarks, REDUCE yields an effective improvement of the WCRT approximation using MaxCY and compared to the existing MaxTC.

7 Conclusions

Synchronous programs react to the environment using discrete instants, called reactions. Worst case reaction time analysis (WCRT) is essential to validate the correctness of the implementation of a program on a given architecture (processor and associated memory hierarchy). Precise analysis requires the elimination of infeasible control-flow paths arising from infeasible state combinations from concurrent threads, known as the tick alignment problem (TAP).

This report presents, for the first time, a number theoretic formulation to solve the TAP for synchronous programs. We start by representing synchronous threads using the formal notion of a tick cost automaton (TCA) which defines an adequate abstract timing semantics for synchronous programs. The problem of WCRT analysis is thereby reduced to the formal manipulations of TCAs. We develop a max-plus algebraic transformation to normalise TCAs to a monocyclic form. This transformation facilitates the transformation of TAP to the Maximum Weight Clique Problem (MWCP) by using the well-known Chinese remainder theorem. The WClque algorithm for MWCP provides the optimal solution to TAP. While it is easy to see that the WCRT algorithm based on MWCP is NP-hard, our exploration of the lower bound of TAP is yet unresolved (see the Appendix for an attempt to reduce an instance of MWCP to TAP).

We have also developed several heuristics in order to solve TAP using PTIME algorithms. We have compared the exact methods based on reachability, integer linear programming (ILP) and MWCP with the developed approximation algorithms. Results
reveal that MWCP is superior to ILP, the most efficient and precise of known methods. Lastly, the proposed approximations, while being non-optimal in theory, work well in practice, suggesting that the tick alignment problem on synchronous programs may exhibit polynomial behaviour.

In the future, we will further explore the lower bound complexity of TAP and study further approximations of real-life benchmarks. From our observations, we think that it could be fruitful to concentrate part of our future efforts on the adaption of a MWCP solving algorithm (like WClique) to the specificity of TAP instances arising from synchronous programs.

References


A Reduction of MWCP to m–TAP

We present a reduction of the Maximum Weight Clique Problem to the m–TAP which depends on computing $O(n^2)$ different primes for a graph with $n$ vertices.

We illustrate the technique by way of an example. Consider the graph $G$ on the left in Fig. 11 consisting of vertices $v_1$–$v_4$ and edges $e_1$–$e_4$. We will write $v_i \leftrightarrow_G v_j$ to state that there is an edge between $v_i$ and $v_j$ in $G$. The right of Fig. 11 depicts a TAP consisting of four m-TCAs, $T_G = \{T_1, T_2, T_3, T_4\}$, one for each vertex of $G$. The edges of $G$ are represented in $T_G$ by a solid line connecting the corresponding m-TCAs, while the absence of an edge in $G$ is coded by a dotted line in $T_G$. Let us call the former connected and the latter disconnected. So, $T_1$, $T_2$, $T_4$ are mutually connected, $T_3$ is connected to $T_4$ but $T_1$ and $T_3$ as well as $T_2$ and $T_3$ are disconnected. The connections in $T_G$ are labelled by distinct prime numbers $p_{\{1,2\}} = 2$, $p_{\{2,4\}} = 3$, $p_{\{1,4\}} = 5$, $p_{\{3,4\}} = 7$, $p_{\{1,3\}} = 11$ and $p_{\{2,3\}} = 13$. In addition, we use four other distinct prime numbers $p_1 = 17$, $p_2 = 19$, $p_3 = 23$ and $p_4 = 29$, corresponding to implicit self-loops at nodes $T_1$, $T_2$, $T_3$ and $T_4$ respectively.

![Figure 11: An undirected graph $G$ (left) coded as a tick alignment problem $T_G$ (right).](image)

The cycle length of each m-TCA in $T_G$ is the product of the prime numbers associated with the connections between $T_i$ and all other $T_j$ and the prime number $p_i$:

\[
\begin{align*}
\phi(T_1) &= p_1 \cdot p_{\{1,2\}} \cdot p_{\{1,3\}} \cdot p_{\{1,4\}} = 17 \cdot 2 \cdot 11 \cdot 5 = 1870 \\
\phi(T_2) &= p_2 \cdot p_{\{1,2\}} \cdot p_{\{2,3\}} \cdot p_{\{2,4\}} = 19 \cdot 2 \cdot 13 \cdot 3 = 1482 \\
\phi(T_3) &= p_3 \cdot p_{\{1,3\}} \cdot p_{\{2,3\}} \cdot p_{\{3,4\}} = 23 \cdot 11 \cdot 13 \cdot 7 = 23023 \\
\phi(T_4) &= p_4 \cdot p_{\{3,4\}} \cdot p_{\{1,4\}} \cdot p_{\{2,4\}} = 29 \cdot 7 \cdot 5 \cdot 3 = 3045.
\end{align*}
\]

In general, $\phi(T_i) = p_i \cdot \phi'(T_i)$ where $\phi'(T_i) \equiv \prod_{j \neq i} p_{\{i,j\}}$. Thus, the greatest common divisors of the cycle lengths correspond to the prime numbers connecting the two m-TCAs in $T_G$, i.e., $gcd(\phi(T_i), \phi(T_j)) = p_{\{i,j\}}$.

Next we choose the tick costs for each m-TCA in a one-hot fashion such that $T_i(k_i) = 1$ for exactly one transition index $0 < k_i < \phi(T_i)$ and $T_i(x) = 0$ for all other $x \neq k_i$ and $0 \leq x < \phi(T_i)$. In other words, the m-TCA $T_i$ has a tick cost of 0 for all transitions except the transition indexed by $k_i$, for which the tick cost is 1. We call these tick offsets $k_i > 0$ the active mode of the $T_i$. The active modes are chosen in such a way that

\[
\begin{align*}
k_i &\equiv_{p_i} 1 & (16) \\
k_i &\equiv_{p_{\{i,j\}}} k_j \Leftrightarrow v_i \leftrightarrow_G v_j & (17)
\end{align*}
\]

A strategy to assign the active modes according to (16) and (17) is to start with any set of nodes forming a clique\(^7\), say the three mutually connected m-TCAs $\{T_1, T_2, T_4\}$ and

\(^7\)This could be a single node, so we do not need to find a clique for this construction.
assign the same active mode $k_1 = k_2 = k_4 = 1$ to all of them, thereby satisfying both (16) and (17). It remains to find a suitable $k_3$ such that $k_3 \mod p_{(3,4)} = k_4 \mod p_{(3,4)} = 1$ (as $T_3$ and $T_4$ are connected) and at the same time $k_3 \mod p_{(2,3)} \neq 1 = k_2 \mod p_{(2,3)}$ and $k_3 \mod p_{(1,3)} \neq 1 = k_1 \mod p_{(1,3)}$ because $T_3$ is not connected to either $T_1$ or $T_2$. A canonical way to achieve this is to put $k_3 \mod p_{(1,3)} = 0$, $k_3 \mod p_{(2,3)} = 0$ and $k_3 \mod p_3 = 1$. Since $p_3, p_{(3,4)}, p_{(1,3)}, p_{(2,3)}$ are distinct prime numbers, by the CRT, the constraints on $k_3$ can be solved uniquely in the range $0 \leq k_3 < \phi(T_3)$. The solution is $k_3 = 21736$.

At this point we have constructed $T_G = \{T_1, T_2, T_3, T_4\}$ in which each m-TCA $T_i$ has tick cost 1 in the active mode $0 \leq k_i < \phi(T_i)$ and 0 otherwise. This means that the cost $T_G(k)$ of $T$ in any tick $k$ can be at most 4. More precisely, if $T_G(k) = d$ then exactly $d$ of the m-TCA has reached their active mode at tick $k$. Formally, there is a set of distinct indices $C \subseteq \{1, 2, 3, 4\}$ such that $|C| = d$ and $k \mod \phi(T_i) = k_i$ for all $i \in C$. This implies by Prop. 4.1 that for all $i, j \in C$, $k_i \mod p_{(i,j)} = k_j \mod p_{(i,j)}$. Thus, by (17), all m-TCA in $C$ are connected, i.e., $v_i \leftrightarrow v_j$ for all $i, j \in C$. This is the same as saying that the set of nodes $V = \{v_i \mid i \in C\}$ is a clique in the graph $G$ of Fig. 11. This shows that at every tick when the TCA $T_G$ produces a joint cost of $d$ we have identified a clique of size $d$ in the graph $G$.

For instance, $T_G(6583291) = 2$ arising from $T_3$ and $T_4$ reaching their active mode because $6583291 \mod 23023 = 21736 = k_3$, $6583291 \mod 3045 = 1 = k_4$ while $T_1$ and $T_2$ are outside of their active mode, which is verified by calculating $6583291 \mod 1870 = 891 \neq 1 = k_1$ and $6583291 \mod 1482 = 247 \neq 1 = k_2$. Hence, the tick count 6583291 activates the clique $\{T_3, T_4\}$. The clique $\{T_1, T_2, T_4\}$ is active at tick count $k = 1$, $T_G(1) = 3$ while $T_3$ is not active since $k_3 = 21736 \neq 1$.

The other direction holds, too. Let $C \subseteq \{1, 2, 3, 4\}$ be the set of indices of a clique $\{v_i \mid i \in C\}$ in the graph $G$. We claim that there is a tick count $k$ such that $T_G(k) = d$ where $d = |C|$ and moreover that the $d$ m-TCA in $C$ are connected, because for all $i, j \in C$, $k_i \mod p_{(i,j)} = k_j \mod p_{(i,j)}$. That is, the clique $\{T_i \mid i \in C\}$ is active at tick cost $k = 1$, so we have found a tick count such that the tick cost of $T_G$ is precisely the size of the clique $C$.

This shows that deciding if $G$ has a maximum weight clique of size $d$ is equivalent to deciding if $\text{wcr}(T_G) = d$. Modulo the generation of $O(n^2)$ distinct prime numbers, this would seem to show that m-TAP is at least as hard as the maximum weight clique problem. However, it is not known if we can such distinct primes in polynomial time. Certainly, generating the first $n$ primes is unlikely to be polynomial since this would imply that the prime factorization problem of RSA cryptography is polynomial which is believed to be even probabilistic polynomial time intractable. Even if we could get hold of $O(n^2)$ distinct primes the explicit size of the generated m-TAP is $O(n \Pi_{i,j} p_{(i,j)})$ and hence exponential. So, our reduction does not prove m-TAP is NP-hard. For us at least, this remains an open question.
<table>
<thead>
<tr>
<th>Nr.</th>
<th>Jahr</th>
<th>Autorin/Autor(en)</th>
<th>Thema</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1989</td>
<td>Augsburger W., Bartmann D., Sinz E.J.</td>
<td>Das Bamberger Modell: Der Diplom-Studiengang Wirtschaftsinformatik an der Universität Bamberg (Nachdruck Dez. 1990)</td>
</tr>
<tr>
<td>2</td>
<td>1990</td>
<td>Esswein W.</td>
<td>Definition, Implementierung und Einsatz einer kompatiblen Datenbankschnittstelle für PROLOG</td>
</tr>
<tr>
<td>3</td>
<td>1990</td>
<td>Augsburger W., Rieder H., Schwab J.</td>
<td>Endbenutzerorientierte Informationsgewinnung aus numerischen Daten am Beispiel von Unternehmenskennzahlen</td>
</tr>
<tr>
<td>4</td>
<td>1990</td>
<td>Ferstl O.K., Sinz E.J.</td>
<td>Objektmodellierung betrieblicher Informationsmodelle im Semantischen Objektmodell (SOM) (Nachdruck Nov. 1990)</td>
</tr>
<tr>
<td>5</td>
<td>1990</td>
<td>Ferstl O.K., Sinz E.J.</td>
<td>Ein Vorgehensmodell zur Objektmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM)</td>
</tr>
<tr>
<td>6</td>
<td>1991</td>
<td>Augsburger W., Rieder H., Schwab J.</td>
<td>Systemtheoretische Repräsentation von Strukturen und Bewertungsfunktionen über zeitabhängigen betrieblichen numerischen Daten</td>
</tr>
<tr>
<td>8</td>
<td>1991</td>
<td>Schwab J.</td>
<td>Ein computergestütztes Modellierungssystem zur Kennzahlenbewertung</td>
</tr>
<tr>
<td>10</td>
<td>1992</td>
<td>Sinz E.J.</td>
<td>Datenmodellierung im Strukturierten Entity-Relationship-Modell (SERM)</td>
</tr>
<tr>
<td>11</td>
<td>1992</td>
<td>Ferstl O.K., Sinz E. J.</td>
<td>Glossar zum Begriffssystem des Semantischen Objektmodells</td>
</tr>
<tr>
<td>13</td>
<td>1992</td>
<td>Esswein W., Locarek H.</td>
<td>Objektorientierte Programmierung mit dem Objekt-Rollenmodell</td>
</tr>
<tr>
<td>14</td>
<td>1992</td>
<td>Esswein W.</td>
<td>Das Rollenmodell der Organisatio: Die Berücksichtigung aufbauorganisatorische Regelungen in Unternehmensmodellen</td>
</tr>
<tr>
<td>16</td>
<td>1992</td>
<td>Schwab K.</td>
<td>Die Implementierung eines relationalen DBMS nach dem Client/Server-Prinzip</td>
</tr>
</tbody>
</table>


Nr. 23 (1994)  Ferstl O.K., Hagemann U.: Simulation hierarchischer objekt- und transaktionsorientierter Modelle


Nr. 31 (1995)  Ferstl O.K., Mannmeusel T.: Gestaltung industrieller Geschäftsprozesse


Nr. 34 (1995)  Sinz E.J.: Ansätze zur fachlichen Modellierung betrieblicher Informationssysteme - Entwicklung, aktueller Stand und Trends -


Applications. (DEXA 98), Vienna, Austria, Aug. 98, pp. 240-249, Springer, Lecture Notes in Computer Sciences, No. 1460


Änderung des Titels der Schriftenreihe Bamberger Beiträge zur Wirtschaftsinformatik in Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik ab Nr. 61

Note: The title of our technical report series has been changed from Bamberger Beiträge zur Wirtschaftsinformatik to Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik starting with TR No. 61
| Nr. 65 (2006) | Schönberger, Andreas: Modelling and Validating Business Collaborations: A Case Study on RosettaNet |
| Nr. 73 (2007) | Martin Schissler: out of print |


<table>
<thead>
<tr>
<th>Nr.</th>
<th>Year</th>
<th>Authors/Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>89</td>
<td>2011</td>
<td>Andreas Henrich, Christoph Schlieder, Ute Schmid [eds.]: Visibility in Information Spaces and in Geographic Environments – Post-Proceedings of the KI’11 Workshop. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 89, Bamberg University, December 2011. ISSN 0937-3349.</td>
</tr>
</tbody>
</table>