A Caching Scheme to Accelerate Kinetic Monte Carlo Simulations of Catalytic Reactions

Srikanth Ravipati,[†] Mayeul d'Avezac,[‡] Jens Nielsen,[‡] James Hetherington,[‡] and Michail Stamatakis[*,†]

[†]*Thomas Young Centre and Department of Chemical Engineering, University College London, Torrington Place, London WC1E 7JE, United Kingdom*

[‡]*Research Software Development Group, Research IT Services, University College London, Torrington Place, London WC1E 6BT, United Kingdom*

E-mail: m.stamatakis@ucl.ac.uk

Abstract

Kinetic Monte Carlo (KMC) simulations have been instrumental in advancing our fundamental understanding of heterogeneously catalyzed reactions, with particular emphasis on structure sensitivity, ensemble effects, and the interplay between adlayer structure and adsorbate-adsorbate lateral interactions in shaping the observed kinetics. Yet, the computational cost of KMC remains high, thereby motivating the development of acceleration schemes that would improve the simulation efficiency. We present an exact such scheme, which implements a caching algorithm along with shared-memory parallelization to improve the computational performance of simulations incorporating long-range adsorbate-adsorbate lateral interactions. This scheme is based on caching information about the energetic interaction patterns associated with the products of each possible lattice process (adsorption, desorption, reaction etc). Thus, every time a reaction occurs ("ongoing reaction") it enables fast updates of the rate constants of "affected reactions", i.e. possible reactions in the region of influence of the "ongoing reaction". Benchmarks on KMC simulations of NO_x oxidation/reduction, yield acceleration factors of up to 20$\times$ when comparing single-thread runs without caching to runs on 16 threads with caching, for simulations with a cluster expansion Hamiltonian that incorporates up to 8th nearest-neighbor interactions.

1 Introduction

Over the last two decades, first-principles based kinetic Monte Carlo (KMC) simulations have gained significant popularity in the computational catalysis field, as means to unravel the striking complexity of heterogeneously catalyzed reactions.[1–5] KMC has thus been instrumental in elucidating various factors that shape the observed kinetics, in particular structure sensitivity,[6–8] ensemble effects,[9] and the interplay between ad-layer structure and adsorbate-adsorbate lateral interactions,[10] in a variety of chemical systems. In KMC simulations essentially one executes a series of reaction events, using information about the statistics of transitions from initial (reactant) states to final (product) states, and obtains a trajectory of configurations. Any observable of interest (e.g. activity, selectivity etc.) is calculated as an average from the resultant trajectory. Hence, KMC focuses on the timescale of reaction events, making use of coarse-grained information about the statistics of reactive paths; this information is captured by a kinetic rate constant that can be computed from first-principles. This coarse-graining makes it possible to sample over billions of reaction events while respecting the physical mechanisms of such transitions (assuming of course that the system simulated is amenable to a KMC representation).[11]

Yet, the computational cost of KMC simulations remains high, in any case much higher than deterministic mean-field equations. However, since the latter employ simplistic approximations, they are unsuitable for modelling systems with strong spatial correlations,[12] for which the KMC framework can be successfully used. In this regard, it is highly desirable to develop acceleration schemes that would improve the computational efficiency of KMC simulations.

To this end, a number of approximate algorithms have been developed, which work by scaling down the rate constants of fast and quasi-equilibrated events under the assumption that this would incur only small errors in the observables.[13–16] These aim at essentially reducing the stiffness of the mathematical problem; however, Andersen et al.[17] showed that such accelerated schemes may lead to large errors when simulating systems whereby independent quasi-equilibrated reactions produce low-coverage species which together participate in subsequent reactions.[17] A different (still approximate) approach based on the theory of absorbing Markov chains, that does not require quasi-equilibration of the events, has also been developed by Pedersen et al.[18], Chill et al.[19], Andersen et al.[20].

On the other hand, exact schemes that reduce the computational burden or distribute it to multiple threads have also been developed and implemented. For instance, the use of inverted lists for fast detection of reaction patterns,[11,21,22] or binary trees for efficient search of the most imminent reaction event,[23] have been shown to significantly accelerate KMC simulations. When modelling systems with long-range adsorbate-adsorbate lateral interactions, shared-memory parallelization can significantly speed up the updates of environment-dependent rate constants after a reaction event.[24] Recently, Hess[25] proposed the use of supercluster, subtraction and supersite algorithms to efficiently account for the changes in lateral interactions during KMC simulations. Such schemes introduce no error in the simulation results, but may incur memory overheads due to the extra amount of information that needs to be stored.

Motivated by the need to further improve simulation efficiency, we have developed a novel scheme that implements caching to accelerate the Graph-Theoretical KMC (GT-KMC) approach.[24] This scheme is intended to accelerate KMC simulations of catalytic systems with complex models of adsorbate lateral interactions, potentially involving long-range terms. In the latter case, each executed reaction modifies the energetics within an extended neighborhood of the lattice, requiring appropriate updates of the rate constants of possible reactions that are in the range of influence of just happened reaction. Such update operations have been found to be the bottleneck of GT-KMC, and shared-memory parallelization was previously implemented, resulting in speed-up factors up to 10× for a benchmark system involving NO_x oxidation/reduction chemistries. The caching scheme we present in this work reduces the number of repetitive detections of energetic interaction pattern instances during such update operations, thereby further improving the efficiency. For the most challenging cluster expansions of the aforementioned benchmark system, we obtain a speedup of 20× when the KMC simulation is run with shared-memory parallelization and caching scheme enabled, compared to the runs using single thread and without caching.

The rest of the paper is organized as follows: in Section 2: Methodology, we explain the technical details behind the caching scheme; in Section 3: Results and Discussion, we present and discuss our benchmark calculations; finally, in Section 4: Summary and Conclusion, we close with a brief overview of the scope and the contribution of our work.

2 Methodology

2.1 Overview of graph-theoretical KMC algorithm

The aim of the KMC algorithm is to estimate catalytic properties of a material (e.g. activity or selectivity), by generating a temporal sequence of snapshots of the catalytic surface through the execution of user-defined events, such as adsorption, desorption, reaction and diffusion. The frequencies of these events are prescribed by appropriately computed kinetic rate constants (e.g. from *ab initio* methods). The observed quantities of interest are calculated as statistical averages, e.g. the turnover frequency is obtained by counting the number of reactant molecules converted per site per time. Key elements of a KMC simulation are presented below, and will feed into our detailed explanations of the novel caching scheme.

2.1.1 Initialization

The KMC algorithm is initialized with:

- the simulation conditions: temperature, T; pressure, P; and gas phase composition, given as the molar fractions of gas species, $\mathcal{Y}$;
- a lattice, representing the catalytic surface: the GT-KMC uses a graph structure to capture the lattice, with its vertices corresponding to sites and its edges defining the neighboring relations among sites;
- the energetics model: a cluster expansion containing a set of coverage patterns, also known as figures, and the energetic contributions thereof, also known as the effective cluster interactions;
- the reaction mechanism: a set of coverage patterns for initial and final states (reactants/products) of an elementary event, e.g. adsorption, desorption, surface reaction, or diffusional hop.

2.1.2 Calculation of coverage-dependent rates

The reaction rate constant of an event in the reaction mechanism is obtained using the Eyring equation:[26]

$$k_{\text{TST}} = \frac{k_B T}{h} \frac{Q^{\ddagger}}{Q_{\text{reactants}}} \exp\left[-\frac{E^{\ddagger}(\boldsymbol{\sigma})}{k_B T}\right] \tag{1}$$

k_{TST} is the rate from transition state theory, k_B is Boltzmann's constant, T is the temperature, h is Planck's constant, $Q^{\ddagger}$ and $Q_{\text{reactants}}$ are the quasi-partition functions for the transition state and the reactants, and $E^{\ddagger}(\boldsymbol{\sigma})$ is the activation energy of the event and is computed via a BEP equation in a consistent way for reversible events.[27,28]

Thus, for a reversible reaction k, the activation energy of the forward event is modeled as:

$$E_{\text{fwd}}^{\text{k},\ddagger} = \max\Big(0, \Delta E_{\text{rxn}}^{\text{k}}(\boldsymbol{\sigma}), E_{\text{fwd},0}^{\text{k}\ddagger} + \omega\left(\Delta E_{\text{rxn}}^{\text{k}}(\boldsymbol{\sigma}) - \Delta E_{\text{rxn},0}^{\text{k}}\right)\Big) \tag{2}$$

where $E^{\mathrm{k},\ddagger}_{\mathrm{fwd},0}$ and $\Delta E^{\mathrm{k}}_{\mathrm{rxn},0}(\boldsymbol{\sigma})$ are the activation barrier for the forward event and the energy of the reaction k at the zero-coverage limit, and ω is the proximity factor.[29] For the corresponding reverse event, the activation energy is modeled as:

$$E^{\mathrm{k},\ddagger}_{\mathrm{rev}} = \max\left(-\Delta E^{\mathrm{k}}_{\mathrm{rxn}}(\boldsymbol{\sigma}), 0, E^{\mathrm{k}\ddagger}_{\mathrm{rev},0} - (1-\omega)\left(\Delta E^{\mathrm{k}}_{\mathrm{rxn}}(\boldsymbol{\sigma}) - \Delta E^{\mathrm{k}}_{\mathrm{rxn},0}\right)\right) \tag{3}$$

where $E^{\mathrm{k},\ddagger}_{\mathrm{rev},0}$ is the activation barrier for the reverse event at the zero coverage limit and also

$$E^{\mathrm{k},\ddagger}_{\mathrm{rev},0} = E^{\mathrm{k},\ddagger}_{\mathrm{fwd},0} - \Delta E^{\mathrm{k}}_{\mathrm{rxn},0} \tag{4}$$

such that the energy of reaction satisfies:

$$\Delta E^{\mathrm{k}}_{\mathrm{rxn}}(\boldsymbol{\sigma}) = E^{\mathrm{k},\ddagger}_{\mathrm{fwd}} - E^{\mathrm{k},\ddagger}_{\mathrm{rev}} \tag{5}$$

Thus, the calculation of activation energies requires a model for the reaction energy; the latter is obtained from the cluster expansion Hamiltonian, $\mathcal{H}$, as follows:

$$\Delta E^{\mathrm{k}}_{\mathrm{rxn}}(\boldsymbol{\sigma}) = \mathcal{H}\left(\boldsymbol{\sigma}'\left(\boldsymbol{\sigma}, \mathrm{k}\right)\right) - \mathcal{H}\left(\boldsymbol{\sigma}\right) + \Delta E^{\mathrm{k}}_{\mathrm{gas}} \tag{6}$$

where $\boldsymbol{\sigma}$ and $\boldsymbol{\sigma}'$ refer to the coverage patterns corresponding to initial and final states of the lattice, and ΔE_{gas} is the change in the energy of gas species participating in an event of reaction k. The total energy of the system is obtained via the cluster expansion Hamiltonian as a sum of contributions from coverage patterns that can be detected on the lattice. These can be single body terms, pair interactions or patterns involving three or more bodies. Formally:

$$\mathcal{H}\left(\boldsymbol{\sigma}\right) = \sum_{p=1}^{N_c} \left(\frac{\mathrm{NCE}_p\left(\boldsymbol{\sigma}\right)}{\mathrm{GM}_p}\right) \times \mathrm{ECI}_p \tag{7}$$

where, ECI_p, GM_p and NCE_p are the effective cluster interaction, graph-multiplicity ($>$ 1 for symmetric patterns) and number of instances (in a given lattice configuration) of pattern p, respectively.

2.1.3 Graph representations and handling of energetic interactions

The main constituents of the KMC simulation, lattice structure, energetic and reaction patterns, are represented as graphs.[24] This representation provides versatility in describing complex reaction events that may involve multiple catalytic sites or adsorbates that bind to more than one site (multi-dentate adsorbates). An example of three energetic interaction patterns and a lattice is shown in fig. 1. For simplicity's sake we consider only a monodentate adsorbate on a lattice that has only one type of site; however, our algorithms are applicable in the general case. Open circles denote empty sites, whereas filled circles denote sites occupied by an adsorbate. The numbers inside the circles are the indices of sites in the lattice or in the energetic pattern, and the labels e_1, e_2 etc. below the pattern sites (of energetic interactions or elementary events) are indexing the entities (adsorbates or empty sites) participating in the pattern. For instance, in the initial state of the forward diffusion event, entity 1 is a "blue" adsorbate, whereas entity 2 is an empty site.

At all times, the GT-KMC algorithm keeps a complete list of energetic interaction patterns, whose contributions make up the total energy of the current lattice configuration. Considering the energetic patterns in our example (fig. 1), there are five instances of the SB energetic pattern in the depicted lattice configuration; $\textcircled{9}$, $\textcircled{15}$, $\textcircled{21}$, $\textcircled{26}$ and $\textcircled{29}$. Regarding 1NN energetic pattern, a pattern detection algorithm (solving the subgraph isomorphism problem)[22,24] identifies the following six instances of the pattern from the given lattice configuration; $\textcircled{9}$-$\textcircled{15}$, $\textcircled{15}$-$\textcircled{21}$, $\textcircled{21}$-$\textcircled{26}$, $\textcircled{15}$-$\textcircled{9}$, $\textcircled{21}$-$\textcircled{15}$ and $\textcircled{26}$-$\textcircled{21}$. However, because of the symmetric nature of this pattern, a distinct pair-wise interaction is double-counted, for example $\textcircled{9}$-$\textcircled{15}$ and $\textcircled{15}$-$\textcircled{9}$ are obtained by first mapping either lattice site $\textcircled{9}$ or $\textcircled{15}$ to pattern site $\textcircled{1}$ and then mapping the other lattice site to pattern site $\textcircled{2}$. To correct for this over-counting, the graph-multiplicity of the pattern is defined as two, and the ECI of the pattern is the energy contribution of a symmetrically distinct 1NN pair of adsorbates.

Finally, the 2NN pattern involves an unspecified site, which can be mapped to a lattice site irrespective of whether the later is occupied or empty. A subtlety of the pattern-matching algorithm of the GT-KMC is that, while it tries to find all the possible lattice sites that can be mapped to a specified pattern site, its behavior for unspecified sites is different. Matching of such sites commences only after all the specified sites have been mapped and it stops when the first successful mapping (of the unspecified site) has been found. This is done to avoid unnecessary computational burden and excessive "proliferation" of pattern instances. Thus, the GT-KMC pattern detection algorithm would identify only one of following two instances for the 2NN pattern: either ⑮-⑳-㉖ or ⑮-㉑-㉖, for which the specified sites are identical. Similarly, only one of the following will be detected: ㉖-⑳-⑮ or ㉖-㉑-⑮. Thus, we have two instances of the 2NN pattern which has a graph-multiplicity of two. Summing all these energetic-interaction contributions, the total energy of the lattice configuration under discussion (fig. 1) is

$$\mathcal{H}\left(\boldsymbol{\sigma}\right) = 5 \times \mathrm{ECI}_{\mathrm{SB}} + \left(\frac{6}{2}\right) \times \mathrm{ECI}_{\mathrm{1NN}} + \left(\frac{2}{2}\right) \times \mathrm{ECI}_{\mathrm{2NN}} \tag{8}$$

2.1.4 Handling of elementary events

As mentioned earlier, reaction patterns (in general: elementary events) are also represented as graphs, and detecting any event requires mapping the sites of a reaction pattern to lattice sites, so that the occupancies and connectivity of the lattice sites match those of the initial state of the event. For example, in fig. 1 adsorption requires finding empty lattice sites (sites ①, ②, ⑦ etc.), desorption requires finding lattice sites that are occupied by the molecule desorbing (sites ⑨, ⑮, ㉑ etc.) and diffusion requires an occupied site that has an empty neighboring site (e.g. ⑨-③, ⑨-④, ⑨-⑧, ..., ⑮-⑩, etc.).

For each newly detected reaction, a rate constant has to be computed which in general depends on the location of spectator species in the neighborhood of the reaction (coverage effects). To this end, the GT-KMC algorithm first computes the energy of the initial state of the reaction by finding all instances of energetic interaction patterns in which the reactants participate, and summing their energy contributions according to eq. (7). This is a fast operation, because the GT-KMC framework keeps a complete and up-to-date list of all energetic interaction terms that make up the total energy of a lattice configuration; thus, the operation just noted, amounts to simply searching through a lookup table. Subsequently, the algorithm makes a *temporary* change in the lattice state, removing the reactants and adding the products of the reaction. It then computes the final state energy of the reaction, which requires detecting new instances of energetic interaction patterns (in which products and spectators, if any, participate) and summing the contributions of these patterns (eq. (7)). Next, the activation energy is calculated from the BEP eq. (2) or eq. (3), for a forward or a reverse event, respectively, and the rate constant follows from eq. (1). Finally, the lattice state is reverted back to that just after the last event took place.

We will now elaborate on the detection of the new energetic patterns (between products and spectators), and will discuss some important algorithmic nuances pertaining to the correctness of pattern counting operations. Since we keep track of the entities that are added to the lattice at all times (adsorbate species or empty sites pseudospecies), the energetic pattern detection algorithm always starts from a molecule known to be located at a certain site of the lattice, and finds all energetic pattern instances in which this molecule participates. We will thus use the expression "an adsorbate is fixed on the pattern" to indicate that a pattern site is already mapped to a lattice site when the pattern detection starts (note that more than one site will be mapped for a multidentate adsorbate). Consequently, calculating the final energy of a reaction could be achieved by the following approach: for each product molecule, loop over the possible energetic interaction patterns that it can participate in (e.g. in fig. 1 a "blue adsorbate" participates to pattern SB as entity e_1, but also to 1NN or 2NN as entity e_1 or e_2). Then, for each such pattern, follow Algorithm 1 which detects the lattice instances of this pattern into which the reaction products participate. The final state energy is a sum of the energetic contributions of these instances for all such patterns and for all the product entities. Extra care has to be taken to avoid double counting if two or more products exist for the given reaction; this is already addressed by Algorithm 1.

To see how this algorithm works, let us consider the lattice shown in fig. 2. Initially, the lattice state contains adsorbates on sites ⑭, ⑯, and ㉑ (highlighted by cyan colour), but then a dissociative adsorption event happens, as part of the KMC simulation. This event results in adding two more adsorbates on the lattice, on sites ⑨ and ⑮ (highlighted by deep blue). Following the algorithm, our *ExcludedAdsorbatesList* contains the adsorbates of sites ⑨ and ⑮.

First let us focus on the 1NN pattern; this defines input variable Π (the pattern whose instances are sought), for which the maximum number of participating entities is two (*max_mmolec*=2). In the first iteration of the loop, line 3 in Algorithm 1, the algorithm fixes the adsorbate of site ⑨, and identifies the pattern instances ⑨-⑭ and ⑨-⑮ in which the fixed adsorbate participates as entity e_1. These pattern instances are added to *ValidPatternInstancesList*. Continuing, the algorithm enters the loop of line 7 and identifies pattern instances ⑭-⑨ and ⑮-⑨ in which fixed adsorbate, adsorbate on site ⑨, participates as entity e_2 (*mmolec* = 2). Yet, it adds only the former to *ValidPatternInstancesList* and rejects the instance ⑮-⑨, because in the latter the adsorbate of site ⑮ which is in *ExcludedAdsorbatesList* appears as entity e_1, i.e. with a smaller entity index than that of the fixed adsorbate (operations in lines 10-15 of the algorithm). As we will see shortly, pattern instance ⑮-⑨ will be counted when the adsorbate of site ⑮ is fixed; thus, the criterion successfully prevents double-counting instances of this pattern.

Now let us consider the second iteration of the loop of line 3. The algorithm fixes the adsorbate of site ⑮ and finds the valid pattern instances corresponding to the 1NN pattern. In this case, the algorithm first identifies pattern instances ⑮-⑭, ⑮-⑨, ⑮-⑯, and ⑮-㉑, in which the fixed adsorbate participates as entity e_1, and adds them to *ValidPatternInstancesList*. Recall that previously, when we had fixed the adsorbate of site ⑨, we rejected the pattern instance ⑮-⑨; this is now detected and added to

ValidPatternInstancesList as needed. Subsequently, the algorithm identifies the pattern instances ⑭-⑮, ⑨-⑮, ⑯-⑮, and ㉑-⑮, and adds to *ValidPatternInstancesList* all of them except ⑨-⑮, the rejection of which follows the same logic as before. In particular, the adsorbate of site ⑨ participates in the pattern with an entity index smaller than that of the fixed adsorbate, and thus it has already been counted when fixing adsorbate ⑨.

Similarly, let us briefly note which pattern instances are detected for the linear triplet pattern ($\Pi \equiv \mathrm{TR}_{\mathrm{linear}}$) when we fix the two newly added adsorbates. By fixing the adsorbate of site ⑨, the algorithm identifies only ⑨-⑮-㉑ as a valid pattern instance in which the fixed adsorbate participates as entity e_1. While looping over entity indices (line 7), it finds the pattern instance ㉑-⑮-⑨ as well but rejects it, because the adsorbate of site ⑮ participates as entity e_2, i.e. with a lower index than that of the fixed adsorbate (e_3). This pattern instance is detected later, in the first iteration of the loop of line 3 when the adsorbate of site ⑮ is fixed.

The procedures we discussed (carried out for every newly detected reaction, as already noted), take place at every KMC step and result in a complete list of new reactions with rate constants that take into account coverage effects. However, these rate constants are computed in the first instance for the lattice configuration for which the new reaction was detected. If some other event happens in the neighborhood of this reaction, it is likely that the rate of the latter will be affected. Thus, update operations need to be carried out, which we discuss in the following section.

2.2 Updates of rate constants after reaction occurrence

During the KMC simulation, whenever an event occurs (referred to as “ongoing reaction”) reactant entities are removed from the lattice, and product entities are added onto the lattice (recall that an entity may be a molecular species or an empty site pseudospecies). In turn, the energetic interaction terms in which reactants participate need to be removed, and new interaction terms, involving the products and possibly spectator molecules, need to be added (see fig. 3). As a result, the change in the lattice configuration due to the "ongoing reaction", may alter the energetics of other adsorbates (spectators) within a local neighborhood, the size of which depends on the interaction range. Since these adsorbates may themselves participate in other reactions (not yet realized), it follows that the activation energies, reaction energies, and rates of these reactions are also altered. We thus refer to these as "affected reactions". It is important to highlight that the latter have not occurred yet, they appear in the list of possible events that can happen in the "future" of the KMC run.

In this section we will discuss two computational schemes that can be used to calculate the updated rate constants of "affected reactions" in the neighborhood of the "ongoing reaction". The first scheme is the one originally implemented in the GT-KMC framework,[24] and relies on the concepts we discussed in section 2.1.4; thus, it will be briefly reviewed, highlighting the need for improving its efficiency. The second scheme is the main methodological contribution of this work. While it is more memory-intensive and entails quite complicated book-keeping, it is significantly more efficient for systems with long-range adsorbate lateral interactions, as we will show later in our benchmarks. Note that in our discussion we use the term "reaction" in a loose way; of course, all the procedures described can be (and are) applied to other types of events as well, e.g. adsorption, desorption or diffusion.

2.2.1 Updates with a single adsorbate fixed when detecting interaction patterns

In the original implementation of the GT-KMC framework, updating the activation energies of "affected reactions" after an "ongoing reaction" happens is done in a brute-force way, using the procedures we already discussed. Algorithm 3 (which uses Algorithm 2 for initialization) shows the pertinent steps: for every "affected reaction", it computes the initial state energy, then temporarily removes the corresponding reactants, adds the products, and computes the final state energy, making use of Algorithm 1, which fixes a single adsorbate in the pattern being detected. Then, the activation energy and the rate constant are computed, and the lattice state is reverted back to that just after the “ongoing reaction” took place.

However, this is computationally wasteful, since, to a large extent it repeats the detection of patterns that have already been detected. Consider for instance the example of fig. 2, with the dissociative adsorption on sites ⑨-⑮ being a possible reaction which has not occurred yet. Let us assume that a diffusion from site ⑯ to ⑩ occurs; this will result in the dissociative adsorption becoming an “affected reaction” whose rate constant has to be updated. While performing the update, one will have to detect patterns between the products and the spectators on sites ⑭ and ㉑. However, these pattern instances were already detected when the dissociative adsorption event was listed as possible reaction (given a lattice state with occupied sites ⑭, ⑯ and ㉑) and are unaltered.

Thus, a potentially more efficient computational scheme could cache the relevant energetic interaction patterns and consider only the necessary changes during rate constant updates. Such a scheme would have to detect patterns in which two entities are fixed, i.e. pairs between the products of the “affected reactions” and the reactants/products of the “ongoing reaction”. The details of this computational scheme are presented in the following section.

2.2.2 Updates using caching and with pairs of adsorbates fixed when detecting interaction patterns

In this scheme, we always store in a cache the counts of energetic interaction patterns, which “make up” the final state energy of a possible reaction. We only count patterns which involve *at least one* newly added entity (among the product molecules of the reaction) and zero or more spectators. For instance, the final state of the dissociative adsorption event considered in fig. 2 entails two SB patterns (⑨, ⑮), ten 1NN patterns of which only five are symmetrically distinct (i.e. ⑨-⑭, ⑨-⑮, ⑭-⑮, ⑮-⑯, ⑮-㉑), four 2NN patterns of which two are symmetrically distinct (⑨-⑮-⑯, ⑭-⑮-㉑), four 3NN patterns of which two are symmetrically distinct (⑨-⑮-㉑, ⑭-⑮-⑯), and finally four TR patterns of which two are symmetrically distinct (same mappings as for the 3NN patterns). Once these counts of interaction patterns are known, computing the reaction energy, activation energy and rate constant of a reaction is trivial (see section 2.1).

Upon occurrence of an “ongoing reaction”, reactants thereof are removed from the lattice and corresponding products are added onto the lattice. Since these reactants and products of the “ongoing reaction” are spectators for the “affected reactions”, the aforementioned counts of interaction patterns have to be updated. To this end, for each “affected reaction”, energy contributions from all the energetic pattern instances involving the products of the “affected reaction” and the reactants of the “ongoing reaction” need to be subtracted, because these reactants are no longer on the lattice (see fig. 3). In a similar way, energy contributions from the energetic pattern instances involving the products of the “affected reaction” and the products of the “ongoing reaction” need to be added (see fig. 3). Note that in both the above cases, we care about products of the “affected reaction”, since we want to evaluate the final state energy thereof. Thus, it is sufficient to find only the energetic pattern instances in which a product of the “affected reaction” and a reactant/product of the “ongoing reaction” participate *together*.

In turn, this necessitates the development of a pattern search algorithm that can accommodate fixing two adsorbates in the pattern, and can correctly “filter out” duplicate detections of the same pattern. Algorithm 4 addresses precisely these needs: given an “affected reaction” and an energetic pattern, Algorithm 4 loops over all pairs of adsorbates ($\mathcal{A}$,$\mathcal{B}$), in which the first adsorbate, $\mathcal{A}$, is a product of the “affected reaction” ($\mathcal{A} \in$ *AffectedExcludedAdsorbatesList* in Algorithm 4) and the second adsorbate, $\mathcal{B}$, is a reactant/product of the “ongoing reaction” ($\mathcal{B} \in$ *OngoingExcludedAdsorbatesList* in Algorithm 4). If, at some iteration, the current pair of adsorbates do not participate together in the given pattern, the iterator effectively moves on to the next pair of adsorbates. If the two adsorbates participate together in the pattern, the algorithm initiates a search for all the lattice instances of that pattern, also considering all the possible combinations of entity numbers for the fixed adsorbates, and adds the instances found to the *TempPatternInstancesList*. An instance in *TempPatternInstancesList* becomes an element of *ValidPatternInstancesList*, only if it satisfies both conditions A and B (lines 9 and 10 in Algorithm 4). These conditions work in a similar way as the condition in line 10 of Algorithm 1, but in this case for a pair of adsorbates, thereby guaranteeing that Algorithm 4 will never return a pattern instance more than once in *ValidPatternInstancesList*. To be clear, note that all the symmetric equivalents of a pattern are still returned correctly, as these are indeed different instances of a pattern, and their over-counting is corrected by the graph-multiplicity.

To better understand the algorithm, let us consider an example lattice configuration as shown in fig. 4. Initially (i.e. before the "ongoing reaction" and the "affected reaction"), the lattice contains only one adsorbate of site ㉒. We will consider the dissociative adsorption on lattice sites ⑭ and ⑳ as the "ongoing reaction", executed at some point in the KMC simulation. Further dissociative adsorption events are possible on many pairs of neighboring lattice sites in the vicinity of ⑭ and ⑳, and all of them will be "affected reactions", since their final state energies are affected by the newly added adsorbates. For our discussion, we consider only one "affected reaction": dissociative adsorption on sites ⑨ and ⑮. Following Algorithm 4 for this example, *AffectedExcludedAdsorbatesList* contains adsorbates on lattice sites ⑨ and ⑮, and *OngoingExcludedAdsorbatesList* contains adsorbates on lattice sites ⑭ and ⑳, and the outermost loops are over pairs, ($\mathcal{A}$,$\mathcal{B}$) of fixed adsorbates, in which $\mathcal{A}$ is a product of the "affected reaction" and $\mathcal{B}$ is a reactant/product of the "ongoing reaction". In this case, the reactants of the latter are "empty-site" pseudo-adsorbates which clearly have no influence in the energetics. We will thus focus our discussion on patterns involving the products of the "ongoing reaction" and products of the "affected reaction".

Let us discuss which instances of 1NN pattern make it to *ValidPatternInstancesList*, according to Algorithm 4. In the first iteration, $\mathcal{A}$ is the adsorbate of site ⑨ and $\mathcal{B}$ is the adsorbate of site ⑭. $\mathcal{A}$ and $\mathcal{B}$ can participate as entity 1 (e_1) and entity 2 (e_2), respectively in the pattern, and also as e_2 and e_1. The pattern detection algorithm has no "preconception" about what the entity number of a molecule in the pattern should be; it tries to find all possible patterns. Thus, ⑨-⑭ and ⑭-⑨ get appended to *TempPatternInstancesList*, respectively for the two different entity mappings just noted. By iterating over all elements in *AffectedExcludedAdsorbatesList* and *OngoingExcludedAdsorbatesList*, the algorithm returns *TempPatternInstancesList* as: ⑨-⑭, ⑭-⑨, ⑮-⑭, ⑭-⑮, ⑮-⑳, ⑳-⑮. All of these instances satisfy both conditions A and B (lines 9, 10 in Algorithm 4), and become elements of *ValidPatternInstancesList*.

Continuing, let us now focus on the 2NN pattern instances that become elements of *ValidPatternInstanceslist* as per Algorithm 4. Again, $\mathcal{A}$ and $\mathcal{B}$ can participate as e_1 and e_2, respectively, as well as e_2 and e_1. When $\mathcal{A}$ and $\mathcal{B}$ are the adsorbates on sites ⑨ and ⑭, respectively, no 2NN pattern instances are detected because of the geometric requirement of this pattern (angle of 120° between the two edges of the pattern). For the next iteration, $\mathcal{A}$ and $\mathcal{B}$ are the adsorbates on sites ⑨ and ⑳, respectively. The algorithm detects and adds to *TempPatternInstancesList* either ⑨-⑭-⑳ or ⑨-⑮-⑳, in which $\mathcal{A}$ and $\mathcal{B}$ participate as e_1 and e_2, respectively (only one of the two patterns is detected because sites with unspecified states are not iterated, as discussed earlier). It is interesting to note that in e.g. the first of these two patterns, lattice site ⑭, which is occupied by an adsorbate in the *OngoingExcludedAdsorbatesList*, is mapped to pattern site ②. However, since the latter has an unspecified state, the adsorbate of site ⑭ is not assigned an entity number in the pattern, and thus, the algorithm does not need to check condition B. In fact, for two body patterns, neither condition A nor B needs to checked; as long as the connectivity, geometry (angles), site types and occupancy of lattice sites[24] agree with those of the pattern for the instance being checked, the latter is accepted as a valid pattern instance. Furthermore, when $\mathcal{A}$ participates as e_2, and $\mathcal{B}$ participates as e_1, either ⑳-⑭-⑨ or ⑳-⑮-⑨ get detected and added to *TempPatternInstancesList*.

To explain how conditions A and B work, let us focus on the $\mathrm{TR}_{\mathrm{bend}}$ pattern, which is a triplet with an angle of 120° between the two edges of the graph, as depicted in Figure 4.

Table 1: List of all the pattern instances of TR energetic pattern that are part of *TempPatternInstancesList* for the example considered in fig. 4 (please see text for more details). Pattern instances represented in gray are only part of *TempPatternInstancesList* and not part of *ValidPatternInstancesList*, while other pattern instances are part of both.

	(⑨,⑭)	(⑨,⑳)	(⑮,⑭)	(⑮,⑳)
(e_1, e_2)	⑨-⑭-⑳			
(e_2, e_1)				⑳-⑮-⑨
(e_2, e_3)				⑨-⑮-⑳
(e_3, e_2)	⑳-⑭-⑨			
(e_1, e_3)		⑨-⑭-⑳		
		⑨-⑮-⑳		
(e_3, e_1)		⑳-⑭-⑨		
		⑳-⑮-⑨		

The lattice instances of this energetic pattern are listed in Table 1. Each column corresponds to one iteration of the double loop (over $\mathcal{A}$ and $\mathcal{B}$) of Algorithm 4. Each row corresponds to the different combinations of entity numbers that can appear upon pattern detection; for example, when the adsorbates of sites ⑨ and ⑭ are fixed in the pattern (first column of Table 1), instance ⑳-⑭-⑨ can be detected, in which the two adsorbates participate as entities e_3 and e_2, respectively (fourth row of Table 1). Instances shown in gray font are only temporarily added to *TempPatternInstancesList* but are never included in *ValidPatternInstancesList*, due to violation of condition A or B (lines 9 and 10) in Algorithm 4. On the other hand, pattern instances shown in black font, are part of both *TempPatternInstancesList* and *ValidPatternInstancesList*. For example, consider the second column and sixth row of Table 1, which lists the instances of TR$_{\text{bend}}$ whereby the adsorbates of sites ⑨ and ⑳ participate as e_3 and e_1, respectively. Instances ⑳-⑭-⑨ and ⑳-⑮-⑨ become elements of *TempPatternInstancesList*, as per line 6 of Algorithm 4. The former pattern instance becomes an element of *ValidPatternInstancesList* as well. However, the latter pattern instance violates condition A (line 9 of Algorithm 4) because the adsorbates on sites $\textcircled{9}$ and $\textcircled{15}$ are part of *AffectedExcludedAdsorbatesList*, and the adsorbate of site $\textcircled{15}$ has lower entity number (e_2) than the fixed adsorbate of site $\textcircled{9}$ (entity number e_3). Thus, the pattern in discussion is discarded in this iteration, in order to avoid duplicate detections (note that this pattern is properly accounted for at a subsequent iteration, as shown in column 4, row 2 of Table 1). Finally, note that at the end of the execution of Algorithm 4, the symmetric pattern instances are correctly accounted for; *ValidPatternInstancesList* contains the instances $\textcircled{9}$-$\textcircled{14}$-$\textcircled{20}$, $\textcircled{20}$-$\textcircled{14}$-$\textcircled{9}$, $\textcircled{9}$-$\textcircled{15}$-$\textcircled{20}$ and $\textcircled{20}$-$\textcircled{15}$-$\textcircled{9}$, and the graph-multiplicity factor makes the necessary corrections to the energy contribution of these patterns, since only two our of these four instances are symmetrically distinct.

To highlight the difference in the efficiency between the two update strategies, using Algorithm 1 versus Algorithm 4, let us go back to fig. 4 and consider the pattern detections necessary to update the rate constant of the "affected reaction". In the "brute force" update strategy, one would temporarily execute the "affected reaction" and use Algorithm 1 to detect interaction patterns that contribute to the final state energy. Then, one would calculate the final state energy, activation energy and finally the rate constant of the reaction. Before the occurrence of the "ongoing reaction" this procedure would detect the 2NN pattern instance $\textcircled{15}$-$\textcircled{16}$-$\textcircled{22}$ (or $\textcircled{15}$-$\textcircled{21}$-$\textcircled{22}$), which contributes to the final state energy. After the occurrence of the "ongoing reaction", Algorithm 1 would detect again the same pattern, in addition to other patterns involving the newly added molecules, i.e. the products of the "ongoing reaction" on sites $\textcircled{14}$ and $\textcircled{20}$. Clearly, this "re-detection" wastes computational resources. Since the adsorbate of site $\textcircled{22}$ was there before and after the "ongoing reaction", it does not contribute to any changes to the activation energy of the "affected reaction". A more efficient strategy, utilising Algorithm 4 would only detect the necessary patterns, i.e. the ones that involve at least one adsorbate among the reactants/products of the "ongoing reaction", and at least one additional adsorbate among the products of the "affected reaction". In our example, the aforementioned 2NN pattern ($\textcircled{15}$-$\textcircled{16}$-$\textcircled{22}$ or $\textcircled{15}$-$\textcircled{21}$-$\textcircled{22}$) would not be detected at all by Algorithm 4 since it involves no reactants/products of the "ongoing reaction". This strategy therefore avoids repetitive detections of patterns that involve already existing (and accounted for) spectators.

Of course, such a strategy requires that we cache information about the previously detected patterns, so as to be able to update this information with minimum computational effort. Thus, in the next section we will describe the cache date-structure along with the procedures it implements.

2.2.3 Cache data-structure

The purpose of the cache data-structure (schematically shown in fig. 5) is to store an up-to-date count of the lattice instances of each energetic pattern that contributes to the final state of a reaction. Such counts have to be cached for each and every possible reaction event that may happen on the lattice. Thus, each of the "leaves" of the structure shown in fig. 5 pertains to one reaction event, and, at a given point in the KMC simulation, there are as many such leaves as the number of possible reactions in the current lattice configuration. For each possible reaction, the data-structure contains two vectors (1-D arrays), χ_0 and χ with size n equal to the number of interaction patterns. The former vector, χ_0, stores the counts of interaction patterns involving only the products of the corresponding reaction; this enables the fast calculation of term $\Delta E^{\mathrm{k}}_{\mathrm{rxn},0}$ in eq. 2-4. On the other hand, vector χ stores the counts of interaction patterns involving products of the reaction and spectator species (if the latter exist), making it possible to quickly calculate term $\Delta E^{\mathrm{k}}_{\mathrm{rxn}}(\boldsymbol{\sigma})$ in eq. 2, 3 and 5.

The cache data-structure implements an addition operation which adds a new "leaf" whenever a new reaction is detected. Invoking Algorithm 1 returns the lattice instances of each energetic interaction pattern involving reaction products; thus, the counts of these instances can be easily added to the data-structure (for χ_0, we require that the instances involve only entities in the *ExcludedAdsorbatesList* in order to be counted). The data-structure also implements a deletion operation which removes an existing "leaf" if the corresponding reaction becomes obsolete due to another event (e.g. adsorption on a previously vacant site that becomes occupied due to a diffusion event).

To keep the cache up-to-date at every step of the KMC simulation, certain update operations have to be performed every time a reaction occurs ("ongoing reaction"). Here, we will discuss the update operation for a single "affected reaction"(steps 4-15 in Algorithm 5); this procedure is repeated for all the "affected reactions" within the neighborhood of the "ongoing reaction". For brevity, we will refer to the reactants of the "ongoing reaction" as "reactants-ongoing", the products of the "affected reaction" as "products-affected", and so on.

At every KMC step, the "reactants-ongoing" are removed from the lattice and the "products-ongoing" are added thereon; thus, when the update operations are performed, the lattice state reflects the initial state of the "affected reaction" and the final state of the "ongoing reaction". Referring back to fig. 3, in order to update the counts of pattern instances in the cache, we need to subtract the counts of interactions between "reactants-ongoing" and "products-affected", and add the counts of interactions between "products-ongoing" and "products-affected". Thus, temporary changes of the lattice state need to be performed. It is important to note that both of these changes happen "behind the scenes" and are not part of the "natural" KMC evolution.

At first, two temporary changes of the lattice state are performed; first one is to execute the "affected reaction" (by removing the "reactants-affected" and adding the "products-affected"; step 4 in Algorithm 5) and the second one is to revert the "ongoing reaction" (by removing the "products-ongoing" and adding the "reactants-ongoing"; step 5 in Algorithm 5). At this point, the lattice state is as if only the affected reaction took place. Now, using Algorithm 4 for each energetic interaction pattern, one can calculate the counts of instances thereof, in which the "products-affected" participate along with the "reactants-ongoing". The counts are subtracted from the appropriate fields of the cache.

Now, the second temporary change previously done on the lattice state is reverted, by removing the reactants and adding the products of the "ongoing reaction" (step 10 in Algorithm 5). At this point, the lattice state is as if both the ongoing and affected reactions took place. Again, using Algorithm 4 one can obtain the counts of lattice instances of an energetic pattern in which the "products-affected" participate along with the "products-ongoing". The counts for each energetic interaction pattern are added to the appropriate fields of the cache data-structure.

Hence, at this point, the cache data-structure contains appropriately updated counts of the pattern instances that are needed to calculate the final state energy of the "affected reaction". At the end of the update procedure, the lattice state is reverted back to the state right after the KMC event was executed, i.e. containing the "products-ongoing" and the "reactants-affected". This is done by also reverting the first temporary change discussed previously (removing the "products-affected" and adding back the "reactants-affected" again). From that point, the "natural" KMC evolution continues.

As an example, Table 2 shows the leaf of the cache data-structure corresponding to the "affected reaction" of fig. 4, before and after the occurrence of the "ongoing reaction". Note that the counts of pattern instances before and after the occurrence of the "ongoing reaction" are the same for the zero-coverage case. Using the cached information, it is easy to compute the energy of the final state of "affected reaction", as a weighted sum of the energies of each of the interaction patterns (the counts times the graph-multiplicity constants are the weighting factors). Moreover, calculating the energy of the initial state of any reaction is always fast, since an up-to-date record of all energetic patterns is maintained throughout the KMC simulation. Thus, appropriate table look-up operations enable the identification of interactions between "reactants-affected" and "products-ongoing", followed by the summation of the corresponding energy contributions to obtain the initial state energy sought.

As a final note in this subsection, we would like to mention that our caching scheme could be made more efficient by caching only the final state energy at the finite coverage and the zero-coverage limit (two real values) for every reaction, and invoking Algorithm 4 for updates. This way, the "makeup" of the final state energy would not be cached, and Algorithm 4 would calculate the counts of instances of two-adsorbate patterns for an "on-the-fly" calculation of the necessary energy updates. Compared to such a scheme, our implementation incurs a memory overhead and a small computational overhead (the latter is small because, in practice, pattern detection via solving subgraph-isomorphism problems[24] is much more computationally intensive than updating the values stored in the cache via integer addition/subtraction). Despite these overheads, our implementation has certain advantages in terms of calculation precision and software sustainability. More specifically, update operations on integer variables (counts of patterns) are not subject to the precision errors expected when updating the reaction energies, which are real variables. Additionally, the counts of the energetic pattern instances at the final state are unique, and this has the advantage that they can be used for thorough debugging, e.g. by comparing the cached values against values computed by Algorithm 1 for a given lattice state during the KMC simulation. On the other hand, calculating the correct difference in the value of the final state energy does not offer a guarantee that the update algorithm is correctly coded, since different combinations of counts may give the same final state energy. For these reasons, we opted for an implementation that is robust and sustainable from a software engineering standpoint, though perhaps suboptimal, mainly in terms of memory footprint.

Table 2: The cache data-structure for the "affected reaction" for the example considered in our discussion, before (left panel) and after (right panel) the occurrence of the "ongoing reaction" (see fig. 4 and text for more details).

	SB	1NN	2NN	TR_{bend}
χ_0	2	2	0	0
χ	2	2	2	0

	SB	1NN	2NN	TR_{bend}
χ_0	2	2	0	0
χ	2	8	4	4

2.3 Shared-memory parallelization

For catalytic systems that exhibit long range interactions, the neighborhood containing "affected reactions" can be quite large, necessitating the update of a large number of leaves of the cache data-structure. These updates can, however, be executed in parallel, by partitioning the set of "affected reactions" to subsets, each of which is assigned to a thread. As long as the temporary lattice changes discussed in the previous section, happen in thread-private mode, any update operations in the cache are safe to execute in parallel, since different reactions correspond to different "leaves" in the cache (which reside in different locations in the memory). Thus, the shared-memory parallelization scheme previously developed, in which the computations on each of the "affected reactions" are distributed across the available threads,[24] can be easily adapted to make use of the caching scheme developed here. This leads to a significant gain in computational efficiency, as discussed in the next section.

3 Results and Discussion

The above procedures and data-structures have been implemented in our KMC software application *Zacros*.[30] The OpenMP[31] framework was adopted for shared-memory parallelization. As a benchmark, we have chosen the NO_x oxidation/reduction on Pt,[24,28] and we assess the performance of the KMC algorithm with caching (making use of the cache data-structure and Algorithm 4 for updates), versus without caching (using Algorithm 1). The following reversible elementary events are modelled (the rate constant for each reaction is shown above or below the corresponding arrow):

Oxidation of NO and reduction of NO_2:

$$NO_{(g)} + O^* \underset{k_{\text{red}}}{\overset{k_{\text{oxi}}}{\rightleftharpoons}} NO_{2(g)} + * \ ,$$

Dissociative adsorption and associative desorption of oxygen:

$$O_{2(g)} + 2* \underset{k_{\text{des}}}{\overset{k_{\text{ads}}}{\rightleftharpoons}} O^* + O^*,$$

Diffusion of oxygen adatoms:

$$\mathrm{O}^* + * \underset{k_{\mathrm{diff}}}{\overset{k_{\mathrm{diff}}}{\rightleftharpoons}} * + \mathrm{O}^*$$

To assess the efficiency of the caching algorithm for cluster expansion Hamiltonians of different complexity, we carried out benchmarks with 3, 5, 8, and 12 figures (terms) in the expansion.[24,28,32] The reader is referred to Figure 2 and Table 1 of Nielsen et al.[24] for more details on the energetics model; here, we briefly note that the 12-figure cluster expansion incorporates up to 8^{th} nearest-neighbor (8NN) interactions. In all simulations the temperature was fixed to 480 K, the partial pressure of O_2 was 0.1 bar, the ratio between the partial pressure of NO_2 versus NO satisfied $log(P_{\mathrm{NO_2}})/P_{\mathrm{NO}}) = -1$, and the lattice size was kept to 18×18 (total of 324 sites). Runs with different numbers of threads were performed to assess the acceleration (speedup) factor with respect to thread count. We have thus defined acceleration factors as ratios between the number of KMC events executed per unit of clock time with/without caching on N threads versus the number of KMC events per unit of clock time without caching on a single thread.

The results of our benchmarks are presented in fig. 6. For simulations considering 3 and 5 figures in the cluster expansion Hamiltonian (5NN and 8NN maximum interaction length, respectively), runs with caching showed only a marginal improvement over runs without. However, for simulations with 8 and 12 figures in the cluster expansion, the performance gains from the caching were found to be significant. In quantitative terms, for simulations that incorporate 12 figures in the cluster expansion Hamiltonian, runs with caching on 16 threads resulted in an acceleration factor of $\approx 20\times$ compared to a single-thread run without caching, and a factor of $\approx 2.5\times$ compared to a run on 16 threads without caching. It is also noteworthy that, for simulations considering 12 figures in the cluster expansion Hamiltonian and run on 8 threads, enabling caching resulted in an acceleration factor of $\approx 3.5-4\times$.

4 Summary and Conclusions

Lattice based kinetic Monte-Carlo simulations are widely used in elucidating molecular level mechanisms in heterogeneous catalysis, and predicting catalytic performance metrics, such as activity and selectivity. An increasing body of literature shows the importance of lateral interactions on the kinetics of catalytic reactions; however, accounting for such effects comes at a significant computational burden. Thus, in practice lateral interactions are often neglected or truncated to short range interactions (typically up to 1^{st} nearest-neighbor. However, such approximations could result in large errors when predicting the behavior of certain systems. In light of this, algorithms and implementations that efficiently account for the effect of lateral interactions in KMC simulations are of utmost importance.

We have developed a parallel caching scheme that addresses this challenge, and reduces the computational burden in the presence of long-range interactions by more than an order of magnitude. The scheme minimizes the computational effort of updating rate constants of reactions when other events happen in the vicinity. In particular, whenever an event (referred to as the "ongoing reaction") happens as part of the KMC propagation, other possible reactions (yet to be realized) in the neighborhood of the "ongoing reaction" are influenced (these are referred to as the "affected reactions"). In the "brute force" implementation of the update procedure, one loops over all "affected reactions", temporarily executes each reaction, detects the interaction patterns between the products and all spectators, computes the difference between initial and final state energies, and invokes a BEP relation to get the new activation energy. However, this is computationally inefficient, since it entails the repetitive detection of interaction patterns with pre-existing spectators (i.e. molecules that did not participate to the "ongoing reaction"). The newly developed scheme, caches the interaction patterns that contribute to the final state energy of each reaction, and uses an algorithm that detects patterns whereby the reactants or products of the "ongoing reaction" participate in tandem with products of the "affected reaction". By this approach, the pattern detection operations are kept to a minimum, and the overall simulation efficiency is improved significantly. As an “added-bonus”, the update operations can be easily parallelized within a shared-memory framework, similar to that previously developed by Stamatakis and co-workers.[24]

This caching approach was benchmarked for a NO_x oxidation/reduction simulation with cluster expansion Hamiltonians containing 3, 5, 8 or 12 terms/figures.[28] The benchmarks for the latter case (the cluster expansion with the longest-range terms, up to 8^{th} nearest-neighbor), yielded acceleration factors of approximately 20× when comparing runs on 16 threads with caching to single-thread runs without caching. The efficacy of the proposed algorithm and its implementation in our software application *Zacros*, is expected to facilitate the wider adoption of high-fidelity models of lateral interactions, towards detailed KMC simulations of complex catalytic systems.

Acknowledgement

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 814416.

The authors also acknowledge the use of the UCL Research Software Development Service (RSD@UCL), as well as the UCL Myriad High Performance Computing Facility (Myriad@UCL) and associated support services in the completion of this work.

References

(1) Sabbe, M. K.; Reyniers, M.-F.; Reuter, K. First-principles kinetic modeling in heterogeneous catalysis: an industrial perspective on best-practice, gaps and needs. *Catal. Sci. Technol.* **2012**, *2*, 2010–2024.

(2) Stamatakis, M. Kinetic modelling of heterogeneous catalytic systems. *J. Phys.: Condens. Matter* **2014**, *27*, 013001.

(3) Stamatakis, M.; Vlachos, D. G. Unraveling the complexity of catalytic reactions via kinetic Monte Carlo simulation: current status and frontiers. *ACS Catal.* **2012**, *2*, 2648–2663.

(4) Salciccioli, M.; Stamatakis, M.; Caratzoulas, S.; Vlachos, D. G. A review of multiscale modeling of metal-catalyzed reactions: Mechanism development for complexity and emergent behavior. *Chem. Eng. Sci.* **2011**, *66*, 4319–4355.

(5) Prats, H.; Illas, F.; Sayós, R. General concepts, assumptions, drawbacks, and misuses in kinetic Monte Carlo and microkinetic modeling simulations applied to computational heterogeneous catalysis. *Int. J. Quantum Chem.* **2018**, *118*, e25518.

(6) Stamatakis, M.; Chen, Y.; Vlachos, D. G. First-principles-based kinetic Monte Carlo simulation of the structure sensitivity of the water–gas shift reaction on platinum surfaces. *J. Phys. Chem. C* **2011**, *115*, 24750–24762.

(7) Nikbin, N.; Austin, N.; Vlachos, D. G.; Stamatakis, M.; Mpourmpakis, G. Catalysis at the sub-nanoscale: complex CO oxidation chemistry on a few Au atoms. *Catal. Sci. Technol.* **2015**, *5*, 134–141.

(8) Prats, H.; Gamallo, P.; Illas, F.; Sayós, R. Comparing the catalytic activity of the water gas shift reaction on Cu (321) and Cu (111) surfaces: Step sites do not always enhance the overall reactivity. *J. Catal.* **2016**, *342*, 75–83.

(9) Mei, D.; Neurock, M.; Smith, C. M. Hydrogenation of acetylene–ethylene mixtures over Pd and Pd–Ag alloys: First-principles-based kinetic Monte Carlo simulations. *J. Catal.* **2009**, *268*, 181–195.

(10) Stamatakis, M.; Piccinin, S. Rationalizing the relation between adlayer structure and observed kinetics in catalysis. *ACS Catal.* **2016**, *6*, 2105–2111.

(11) Jansen, A. P. J. *An introduction to kinetic Monte Carlo simulations of surface reactions*; Springer, 2012; Vol. 856.

(12) Pineda, M.; Stamatakis, M. Beyond mean-field approximations for accurate and computationally efficient models of on-lattice chemical kinetics. *J. Chem. Phys.* **2017**, *147*, 024105.

(13) Núñez, M.; Robie, T.; Vlachos, D. Acceleration and sensitivity analysis of lattice kinetic Monte Carlo simulations using parallel processing and rate constant rescaling. *J. Chem. Phys.* **2017**, *147*, 164103.

(14) Chatterjee, A.; Voter, A. F. Accurate acceleration of kinetic Monte Carlo simulations through the modification of rate constants. *J. Chem. Phys.* **2010**, *132*, 194101.

(15) Danielson, T.; Sutton, J. E.; Hin, C.; Savara, A. SQERTSS: Dynamic rank based throttling of transition probabilities in kinetic Monte Carlo simulations. *Comput. Phys. Commun.* **2017**, *219*, 149–163.

(16) Dybeck, E. C.; Plaisance, C. P.; Neurock, M. Generalized temporal acceleration scheme for kinetic Monte Carlo simulations of surface catalytic processes by scaling the rates of fast reactions. *J. Chem. Theory Comput.* **2017**, *13*, 1525–1538.

(17) Andersen, M.; Plaisance, C. P.; Reuter, K. Assessment of mean-field microkinetic models for CO methanation on stepped metal surfaces using accelerated kinetic Monte Carlo. *J. Chem. Phys.* **2017**, *147*, 152705.

(18) Pedersen, A.; Berthet, J.-C.; Jónsson, H. Simulated annealing with coarse graining and distributed computing. International Workshop on Applied Parallel Computing. 2010; pp 34–44.

(19) Chill, S. T.; Welborn, M.; Terrell, R.; Zhang, L.; Berthet, J.-C.; Pedersen, A.; Jonsson, H.; Henkelman, G. EON: software for long time simulations of atomic scale systems. *Modell. Simul. Mater. Sci. Eng.* **2014**, *22*, 055002.

(20) Andersen, M.; Panosetti, C.; Reuter, K. A Practical Guide to Surface Kinetic Monte Carlo Simulations. *Front. Chem.* **2019**, *7*, 202.

(21) Schulze, T. Kinetic Monte Carlo simulations with minimal searching. *Phys. Rev. E* **2002**, *65*, 036704.

(22) Stamatakis, M.; Vlachos, D. G. A graph-theoretical kinetic Monte Carlo framework for on-lattice chemical kinetics. *J. Chem. Phys.* **2011**, *134*, 214115.

(23) Chatterjee, A.; Vlachos, D. G. An overview of spatial microscopic and accelerated kinetic Monte Carlo methods. *J. Comput.-Aided Mater. Des.* **2007**, *14*, 253–308.

(24) Nielsen, J.; d'Avezac, M.; Hetherington, J.; Stamatakis, M. Parallel kinetic Monte Carlo simulation framework incorporating accurate models of adsorbate lateral interactions. *J. Chem. Phys.* **2013**, *139*, 224706.

(25) Hess, F. Efficient Implementation of Cluster Expansion Models in Surface Kinetic Monte Carlo Simulations with Lateral Interactions: Subtraction Schemes, Supersites, and the Supercluster Contraction. *J. Comput. Chem.* **2019**, *40*, 2664–2676.

(26) Eyring, H. The activated complex in chemical reactions. *J. Chem. Phys.* **1935**, *3*, 107–115.

(27) Wang, S.; Temel, B.; Shen, J.; Jones, G.; Grabow, L.; Studt, F.; Bligaard, T.; Abild-Pedersen, F.; Christensen, C.; Nørskov, J. Universal Brønsted-Evans-Polanyi relations for C-C, C-O, C-N, N-O, N-N, and O-O dissociation reactions. *Catal. Lett.* **2011**, *141*, 370–373.

(28) Wu, C.; Schmidt, D.; Wolverton, C.; Schneider, W. Accurate coverage-dependence incorporated into first-principles kinetic models: Catalytic NO oxidation on Pt (111). *J. Catal.* **2012**, *286*, 88 – 94.

(29) Grabow, L. C.; Gokhale, A. A.; Evans, S. T.; Dumesic, J. A.; Mavrikakis, M. Mechanism of the water gas shift reaction on Pt: First principles, experiments, and microkinetic modeling. *J. Phys. Chem. C* **2008**, *112*, 4608–4617.

(30) *Zacros*: Advanced lattice-KMC made easy. `http://zacros.org`, Accessed on 2020-07-16.

(31) The OpenMP API specification for parallel programming. `https://www.openmp.org`, Accessed on 2020-07-16.

(32) Schmidt, D. J.; Chen, W.; Wolverton, C.; Schneider, W. F. Performance of Cluster Expansions of Coverage-Dependent Adsorption of Atomic Oxygen on Pt(111). *J. Chem. Theory Comput.* **2012**, *8*, 264–273.

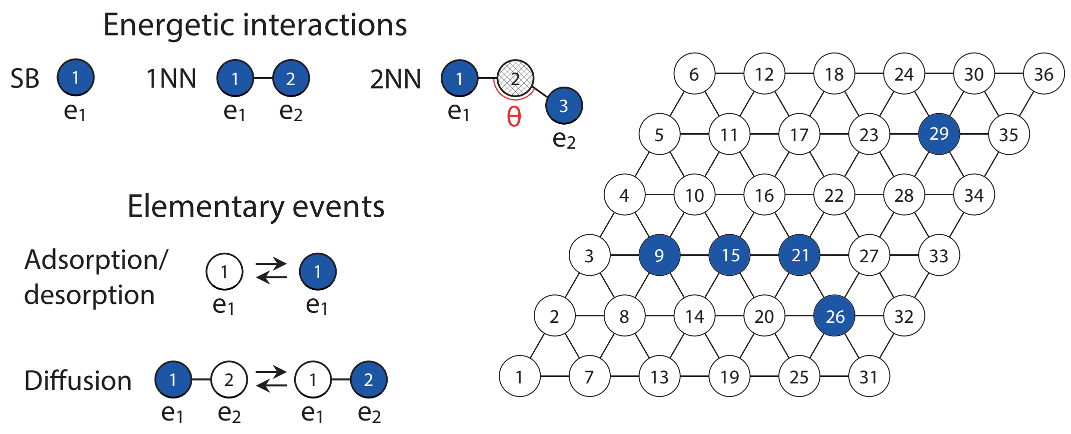


Figure 1: An example showcasing principal constituents of kinetic Monte Carlo simulation; a lattice, energetic patterns and elementary events. Open circles represent empty sites and filled circles represent sites occupied by the "blue adsorbates". Numbers in circles represent site numbers in the lattice, energetic patterns or elementary events. The entities participating in an energetic pattern or an elementary event are labeled as e_1, e_2 etc. In the energetic interactions presented, SB, 1NN, and 2NN refer to single body, first and second nearest-neighbor interactions respectively.

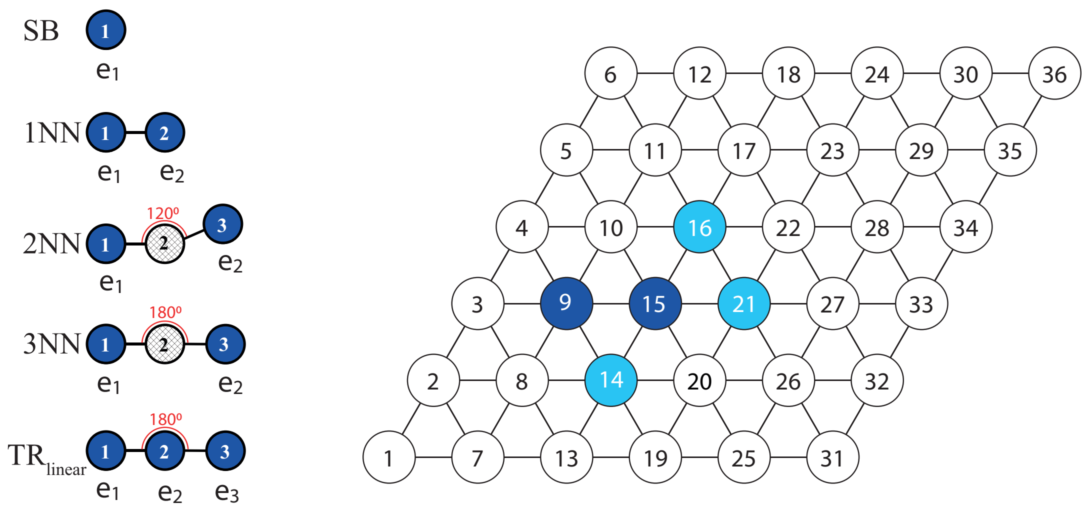


Figure 2: Example of a lattice configuration to aid the discussion of Algorithm 1, which computes the list of energetic patterns contributing to the final state energy by fixing an adsorbate in the interaction pattern. The adsorbates on sites 14, 16, and 21 pre-exist while dissociative adsorption ("ongoing reaction") adds products on sites 9 and 15. The energetic interaction patterns in which these adsorbates participate are shown on the left. TR$_{\text{linear}}$ refers to the linear triplet pattern.

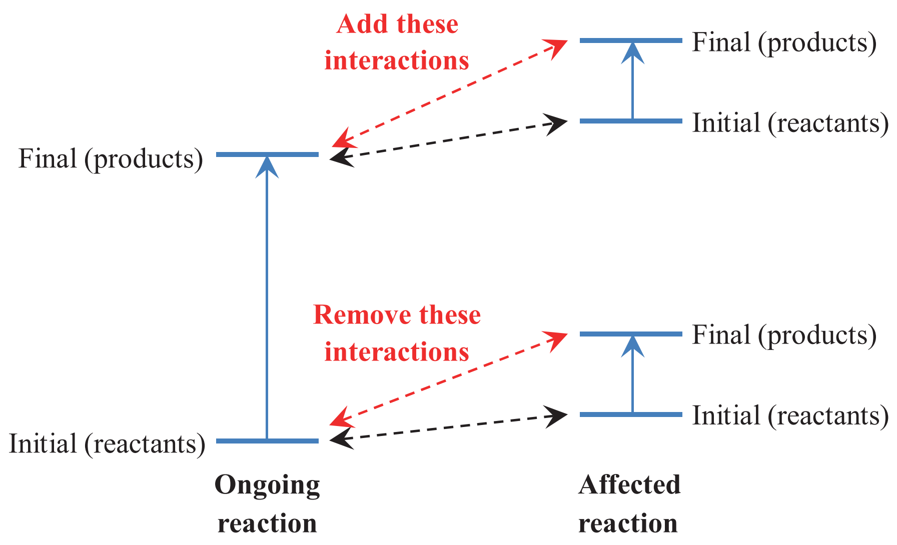


Figure 3: A schematic highlighting the energetics that need to be subtracted/added as a result of an "ongoing reaction", between the products of an "affected reaction" and reactants/products of the "ongoing reaction".

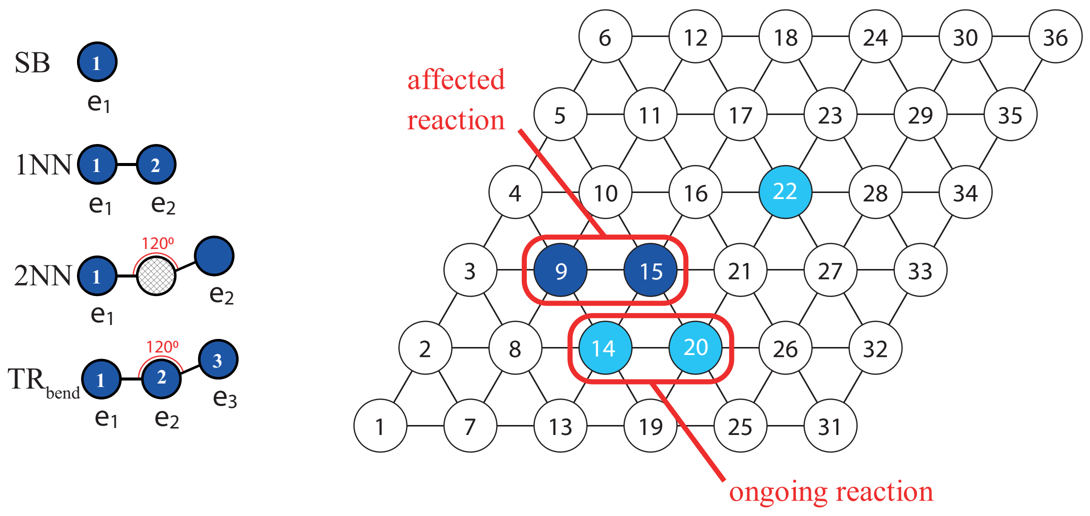


Figure 4: An example lattice configuration to aid in discussing Algorithm 4 that detects pattern instances by fixing a pair of adsorbates in the interaction pattern. Dissociative adsorption on sites 14 and 20 is the “ongoing reaction” and dissociative adsorption on sites 9 and 15 is chosen as the “affected reaction”. TR$_{\text{bend}}$ refers to triplet energetic pattern with an angle constraint of 120° between the two edges of the pattern.

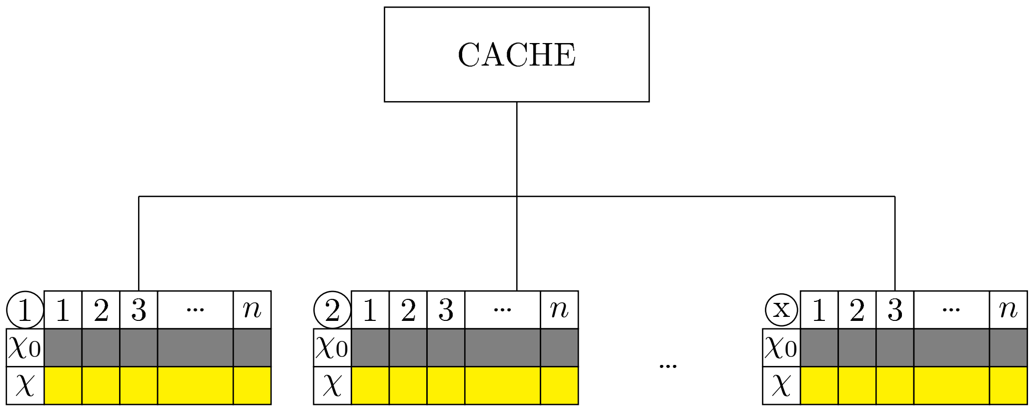
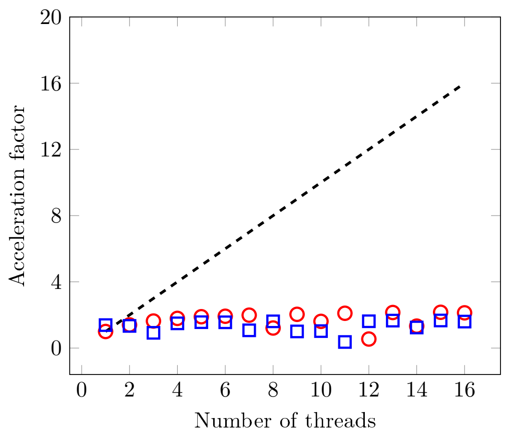
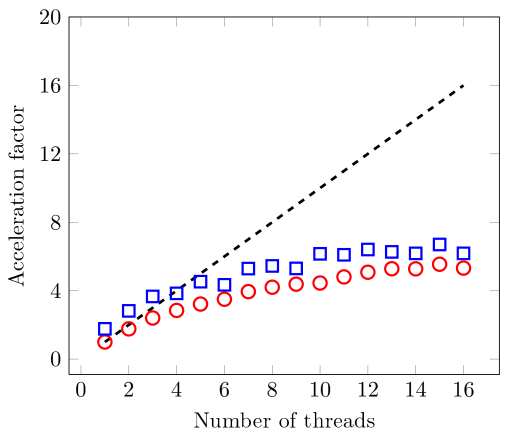


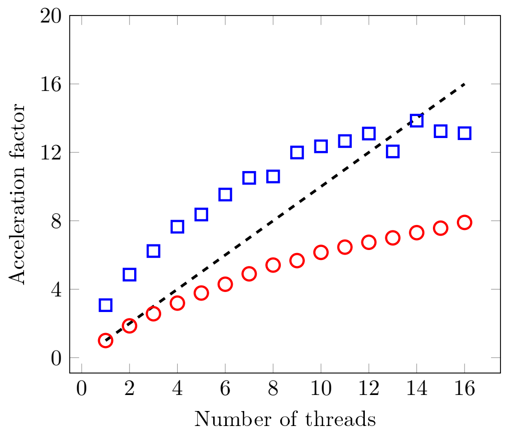
Figure 5: Data-structure used to cache energetics of different patterns in which the products of all possible reactions participate. Encircled 1, 2, . . . , x, are the all possible reactions. 1, 2, 3, . . . n are different energetic patterns in which products of an affected reaction may participate. χ_0 and χ refer to counts of instances of an energetic pattern corresponding to zero-coverage and coverage-dependent cases, respectively.



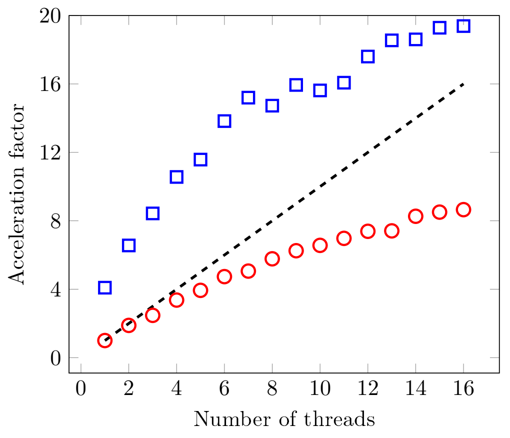
(a) 3-Figure Cluster Expansion



(b) 5-Figure Cluster Expansion



(c) 8-Figure Cluster Expansion



(d) 12-Figure Cluster Expansion

Figure 6: Comparison of performance without and with caching enabled for simulations considering different cluster expansion Hamiltonians of different complexity (with different numbers of figures/terms). Circles and squares correspond to runs without versus with caching, respectively. The dashed line represents the ideal linear scaling with the number of threads used in the absence of caching.

Algorithm 1: Detection of energetic pattern instances involving reaction products

```
Data: $\Pi$: Energetic interaction pattern whose instances on the lattice are sought.
      ExcludedAdsorbatesList: List of all newly added entities (reaction products).
Result: ValidPatternInstancesList: List of all lattice instances of energetic interaction
        pattern $\Pi$, which contribute to the final state energy of a reaction.
1  begin
2      Initialize ValidPatternInstancesList as an empty list;
3      for $\mathcal{A}$ in ExcludedAdsorbatesList do
4          Find all the lattice instances of pattern $\Pi$, in which adsorbate $\mathcal{A}$ participates as
             entity $e_1$ and add them to ValidPatternInstancesList;
5          Set max_mmolec as the maximum number of entities participating in $\Pi$;
6          Initialize TempPatternInstancesList to an empty list;
7          for mmolec = 2,...,max_mmolec do
8              Find all the lattice instances of pattern $\Pi$ in which $\mathcal{A}$ participates as entity
                 $e_{mmolec}$ and add them to TempPatternInstancesList;
9              for $\mathcal{P}$ in TempPatternInstancesList do
10                 condition = no other adsorbate in ExcludedAdsorbatesList participates in $\mathcal{P}$
                     with entity number $<$ mmolec;
11                 if condition then
12                     Accept $\mathcal{P}$ and add it to ValidPatternInstancesList;
13                 else
14                     Reject $\mathcal{P}$ and continue with the next pattern instance in the loop;
15                 end
16             end
17         end
18     end
19 end
```

Algorithm 2: Initializer and populator for Algorithms 3 and 5

```
1 begin
2     Initialize *AffectedSites*, the sites in the neighborhood of *OngoingRxn*, as an empty list;
3     Initialize *AffectedAdsorbates*, the spectators in the neighborhood of *OngoingRxn*, as an
        empty list;
4     Initialize *AffectedRxn* as an empty list;
5     Populate *AffectedSites*: starting from the sites of *OngoingRxn*, list all neighboring sites
        up to depth equal to that of the maximum interaction length;
6     Populate *AffectedAdsorbates* by listing all adsorbates bound (with at least one
        dentate) to any of the sites in *AffectedSites* list;
7     Populate *AffectedRxn* by listing all the reactions in which any of the adsorbates in
        *AffectedAdsorbates* participate;
8 end
```

Algorithm 3: Update of reaction rates in the neighborhood of an ongoing reaction

```
Data: OngoingRxn: Ongoing reaction.
      EnergPatternList: List of all energetic interaction patterns in cluster expansion.
      GlobClusterList: Indexed list of all the energetic pattern instances in which every
      adsorbate participates (the list is indexed by adsorbate), for the current KMC
      state.
Result: AffectedRxn: Vector of affected reactions.
        $k_{\mathrm{TST}}$: Vector of updated rate constants of all affected reactions.
1  begin
2      Invoke Algorithm 2 to initialize and populate necessary lists;
3      for $\mathcal{R}$ in AffectedRxn do
4          Initialize InitialStateValidPatternInstancesList as an empty list;
5          Perform a table lookup in GlobClusterList and collect the pattern instances that
             contribute to the initial state of this reaction;
6          Append the instances just found to InitialStateValidPatternInstancesList;
7          Perform a temporary change to execute the reaction on the lattice;
8          Initialise FinalStateValidPatternInstancesList as an empty list;
9          Populate ExcludedAdsorbatesList with the newly added entities (reaction
             products);
10         for $\Pi$ in EnergPatternList do
11             Execute Algorithm 1 to find pattern instances of $\Pi$ that contribute to the final
                 state of this reaction and add them to FinalStateValidPatternInstancesList;
12         end
13         Now that InitialStateValidPatternInstancesList and
             FinalStateValidPatternInstancesList are known, compute the initial and final
             lattice energy of reaction $\mathcal{R}$ from eq. (7);
14         Calculate the activation energy of reaction $\mathcal{R}$ from eq. (2) or eq. (3) (depending on
             whether reaction $\mathcal{R}$ is a forward or a reverse event, respectively);
15         Calculate the rate constant of reaction $\mathcal{R}$, $k_{\mathrm{TST}}^{\mathcal{R}}$, from eq. (1);
16     end
17 end
```

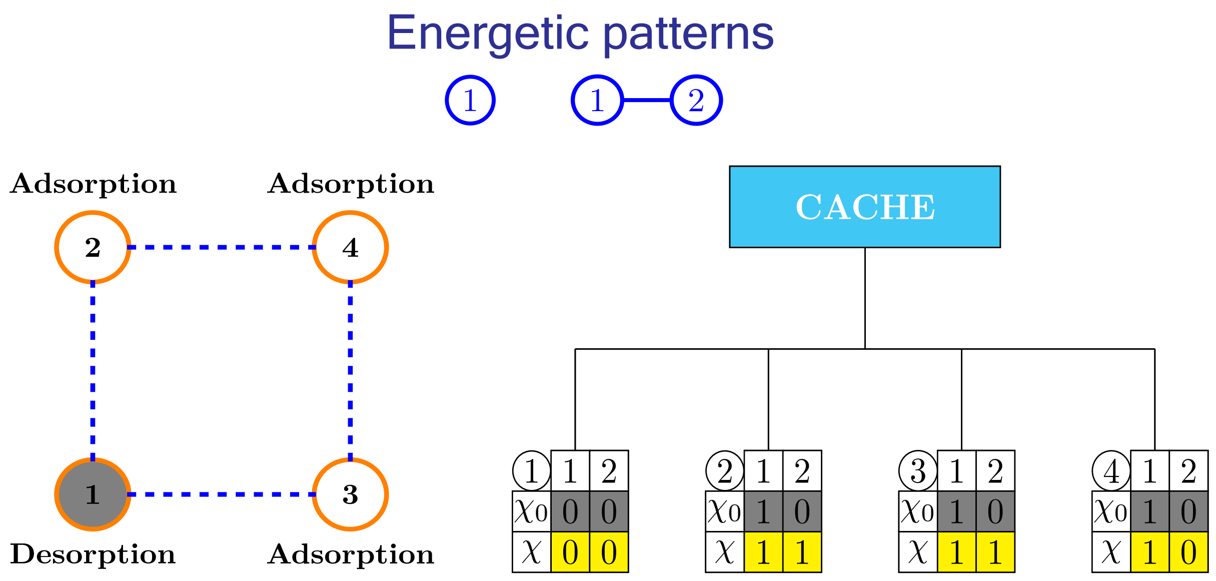
Algorithm 4: Detection of energetic pattern instances by fixing a pair of adsorbates in the pattern

```
Data: Π: Energetic interaction pattern whose instances on the lattice are sought.
      AffectedExcludedAdsorbatesList: List of all products of an affected reaction.
      OngoingExcludedAdsorbatesList: List of all reactants/products of the ongoing
      reaction.
Result: ValidPatternInstancesList: List of all lattice instances of energetic interaction
        pattern Π, which contribute to the final state energy of a reaction.
1  begin
2      Initialize TempPatternInstancesList and ValidPatternInstancesList as empty;
3      for $\mathcal{A}$ in AffectedExcludedAdsorbatesList do
4          for $\mathcal{B}$ in OngoingExcludedAdsorbatesList do
5              if both $\mathcal{A}$ and $\mathcal{B}$ participate in Π then
6                  Find all lattice instances of the pattern and add them to
                     TempPatternInstancesList;
7              end
8              for $\mathcal{P}$ in TempPatternInstancesList do
9                  Condition A = no other adsorbate in AffectedExcludedAdsorbatesList
                     participates in pattern instance $\mathcal{P}$ with entity number less than that of $\mathcal{A}$;
10                 Condition B = no other adsorbate in OngoingExcludedAdsorbatesList
                     participates in pattern instance $\mathcal{P}$ with entity number less than that of $\mathcal{B}$;
11                 if condition A and condition B are both true then
12                     Accept $\mathcal{P}$ and add it to the ValidPatternInstancesList ;
13                 else
14                     Reject $\mathcal{P}$ and continue with the next pattern instance;
15                 end
16             end
17             Reset TempPatternInstancesList as empty list;
18         end
19     end
20 end
```

Algorithm 5: Update of caching structure after an "ongoing reaction"

```
   Data: OngoingRxn: Ongoing reaction.
         EnergPatternList: List of all energetic interaction patterns in cluster expansion.
   Result: Cache: Data-structure that stores the counts of instances of each energetic pattern that
           contributes to the final state of every reaction possible for the current lattice state.
 1 begin
 2     Invoke Algorithm 2 to initialize and populate necessary lists;
 3     for $\mathcal{R}$ in AffectedRxn do
 4         Perform a temporary change to execute the affected reaction on the lattice (i.e. remove the
             reactants of AffectedRxn and add its products to the lattice);
 5         Perform a temporary change to revert the ongoing reaction on the lattice (i.e. remove the
               products of OngoingRxn add its reactants to the lattice) // Now the lattice state
               is as if only AffectedRxn occurred
 6         for $\Pi$ in EnergPatternList do
 7             Use Algorithm 4 to detect the energetic pattern instances containing reactants of
                 OngoingRxn and products of AffectedRxn;
 8             Decrement the numbers of pattern instances of $\Pi$ in Cache (for OngoingRxn) by the
                 count of instances obtained in the previous step;
 9         end
10         Revert the temporary change of step 5, thereby removing the reactants of OngoingRxn and
               add its products again // Now the lattice state is as if both AffectedRxn
               and OngoingRxn occurred
11         for $\Pi$ in EnergPatternList do
12             Use Algorithm 4 to detect the energetic pattern instances containing products of
                 OngoingRxn and products of AffectedRxn;
13             Increment the numbers of pattern instances of $\Pi$ in Cache (for OngoingRxn) by the
                 count of instances obtained in the previous step;
14         end
15         Revert the temporary change of step 4, thereby removing the products of AffectedRxn and
             add its reactants again;
16     end
17 end
```

TOC Graphic


Energetic patterns
1
1 2
Adsorption
Adsorption
2
4
1
3
Desorption
Adsorption
CACHE
1 1 2
χ_0 0 0
χ 0 0
2 1 2
χ_0 1 0
χ 1 1
3 1 2
χ_0 1 0
χ 1 1
4 1 2
χ_0 1 0
χ 1 0