MODELING ORGANORHODIUM CATALYSIS

by

Alexander S. Bayden

BS, Virginia Polytechnic Institute and State University, 2000

Submitted to the Graduate Faculty of

Arts and Sciences in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2005

UNIVERSITY OF PITTSBURGH

FACULTY OF ARTS AND SCIENCES

This dissertation was presented

by

Alexander S. Bayden

It was defended on

November 28, 2005

and approved by

Kenneth D. Jordan, Ph.D., Department of Chemistry

Kay M. Brummond, Ph.D., Department of Chemistry

Christian E. Schafmeister, Ph.D., Department of Chemistry

Daniel M. Zuckerman , Ph.D., Department of Computational Biology

Dissertation Director: Kenneth D. Jordan, PhD, Department of Chemistry

MODELING ORGANORHODIUM CATALYSIS

Alexander S. Bayden, PhD

University of Pittsburgh, 2005

The first project considered in this dissertation was the improvement of an existing global optimization algorithm that uses extended dimensionality to find global minima of Lennard-Jones clusters. The speed of this algorithm was increased by three orders of magnitude, primarily by improving the algorithm for compressing the system from 4D to 3D at constant energy.

The second project was modeling the adsorption of H_2 molecules on the Si(100) surface using density functional theory (DFT) with the PW91 functional. Consistent with the experiments, the calculations predicted an energetic preference for clustering of occupied sites in a dimer row. However, our calculations did not verify the unbuckling induced by H_2 adsorption reported by Buehler and Boland.

The third project was modeling molybdenum and rhodium-catalyzed [2 + 2 + 1] cyclocarbonylation reaction using DFT with the B3LYP functional. We found that in the rhodium-catalyzed [2 + 2 + 1] cyclocarbonylation reactions of allenes the oxidative addition step determined both the rate and the product of the reaction. For the molybdenum-promoted reaction the rate was controlled not by oxidative addition, but by the next step, the attachment of a carbon monoxide molecule from the media to the molybdenum atom.

The fourth project was modeling the transfer of hydrogen from one side of the heterocyclic ring to another in rhodium(I) catalyzed allenic Pauson-Khand type reactions. Our calculations showed that this process occurs after the cyclization step. We have also discovered a novel mechanism for this process - hydride transfer; however, we believe that in most cases the reaction proceeds by β-hydride elimination.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

I would like to express my appreciation of and admiration for the faculty and staff of the Chemistry Department at the University of Pittsburgh. In particular I would like to express my thanks to Professors Kenneth Jordan and Kay Brummond for guiding me through challenging research projects and to Professor Christian Schafmeister for mentoring me concerning my proposal. Additionally I would like to thank Professor Joseph Grabowski for supporting my web programming efforts.

I wish to thank the members of the Jordan group, past and present, for help in science, computer matters and for the congenial atmosphere that pervades the group. I would especially like to thank Jan Steckel for helping me with silicon calculations, Albert DeFusco and Richard Christie for providing great insight into quantum chemistry and computer problems and Ming-Kang Tsai for helping me to understand the administrative requirements for writing and submitting the dissertation. Additionally I would like to thank other Jordan group members: Suzanne Gardner, Wendy Lampart, Kadir Diri, Jun Cui, Valerie McCarthy, Yevgeny Myshakin, Thomas Sommerfeld and Hao Jiang, Tae Hoon Choi, Hanbin Liu, Mahalakshmi Sahasrananam, Ellen Vayner, Feng Wang, Dominic Alfonso, Karen Karapetian and Arnold Tharrington.

I have also benefited greatly from interacting with the Brummond group members: Branko Mitasev, Thomas Painter and Anthony Casarez. Their explanations of synthetic procedures greatly aided me in understanding the mechanisms of the organic reactions studied in this dissertation.

It is hard for me to imagine writing this dissertation without the help of technical writers: Maria Hughes and Steve Kimos.

Finally I would like to dedicate this dissertation to my parents. Their seemingly limitless patience and support made my graduate studies possible.

1. Introduction

1.1. Nanotechnology

1.1.1. Proteins – natural nanodevices

Nanotechnology is a science that studies devices between 1 and 100 nm in size. Long before people were able to build their own nanodevices, nanodevices already existed in nature. For example proteins are nanodevices. They serve various useful purposes in living organisms that include catalyzing reactions and signaling. If one can adjust the behavior of a protein, one can potentially treat a disease. For example, bacteria have a lot of proteins in them. Some bacteria are malignant. A small molecule that can inhibit the functioning of certain proteins in bacteria can be used to treat bacterial infections. Such small molecules are called antibiotics. Chapter 4 deals with the synthesis of a small molecule that can act as an antibiotic agent. Chapter 5 deals with the mechanism of a synthetic step used in producing other small molecules that can regulate protein activity.

1.1.2. Global optimization – a way to find protein structure

To design a small molecule that will either enhance or inhibit protein activity it helps to know the structure of the protein. Today, because of advances in the field of bioinformatics, the sequence of amino acids in the protein is known most of the time. However it is hard to find the structure. By using molecular mechanics with modern force fields it is possible to predict relative energies of protein conformations, however the number of possible conformations for a protein is so vast, that it is impossible to sample all of them exhaustively. Global optimization algorithms have been developed to solve problems like these. They allow one to find the best solution without exhaustively sampling all of the solutions. Chapter 1 deals with improvement of an existing global optimization algorithm.

1.1.3. Man-made nanodevices

Today people can make their own nanodevices like nanotubes and are able to make them perform useful functions. For example carbon nanotubes have been used to build nanoscale transistors.

1.1.4. Interfacing nanodevices to modern electronic devices

A human can not manipulate a nanodevice without appropriate tools. One has to interface them with conventional electronic devices. Most electronic devices today are built from silicon. It is possible to manufacture very intricate silicon-based devices. If one wants to use nanosized molecular electronics devices, one should consider interfacing them to modern silicon-based devices.

A series of experiments showed a promising way to integrate nanotechnology with the existing silicon-based technology. Recent experiments have shown that it may be possible to construct nanostripes on the Si(100) surface by H_2 adsorption. Modeling this process is described in Chapter 3.

1.2. Density Functional Theory

1.2.1. Introduction

Density Functional Theory (DFT)[1,2,3] was first introduced in the 1920s by Fermi and Thomas[4] who suggested that the energy as well as all other electronic properties of a ground-state atom or molecule can be uniquely determined by the electron density, $\rho(\bar{r})$ which replaces the more familiar N-electron wave function, $\Psi(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_N)$. Four decades later, in 1964, Hohenberg and Kohn[5] gave the first formal proof that the ground state energy (E_0) of an electronic system is uniquely identified by its density, thereby validating the use of $\rho(\bar{r})$. They did not show, however, how to calculate E_0 from the ground state electron density. Kohn and Sham demonstrated a year later that E_0 could be determined by using a set of localized one-electron orbitals, called Kohn-Sham (KS) orbitals, with the resulting orbital equations having the same form as Hartree-Fock (HF) equations.[6] But there are distinct differences between the HF and DFT methods, as emphasized by Becke.[7] Most importantly, DFT does include electron correlation effects, giving results comparable to those[8] based on second-order Moller-Plesset (MP2) perturbation theory (except where dispersion dominates). Yet the computational cost of DFT calculations is of the same order of magnitude as (or even less than) HF systems, making it an appealing method for larger systems.

DFT uses the term functional, which can be explained in the following way.[1, 9] A function, f(x), associates a number with each value of the variable x. A functional, F[f(x)], on the other hand, associates a number with each value of the function f(x). For example, the expression

$<\Psi|\hat{H}|\Psi>$ represents a functional because $\hat{H}$ is a function. The resulting value E is said to be a functional of Ψ, or equivalently written, $E[\Psi]$.

The purpose of this chapter is to give a brief overview of DFT as applied to closed-shell systems. The next section will show the connection between DFT and traditional wave equation formalism, followed by a section presenting the Kohn-Sham equations. The last section will discuss exchange-correlation functionals, with the emphasis on the Beck3LYP functional.[17]

1.2.2. Hohenberg and Kohn Theorems

If one uses atomic units the Hamiltonian for an atomic or molecular system can be written as:

$$\hat{H} = -\tfrac{1}{2}\sum_{i}\overline{\nabla}_i^2 + \sum_{i}\upsilon(\bar{r}_i) + \tfrac{1}{2}\sum_{i\neq j}\frac{1}{r_{ij}}$$

Equation 1

where

$$\upsilon(\bar{r}_i) = \sum_{a}\frac{Z_a}{r_{ia}}.$$

Equation 2

The first term of **Equation 1** represents the kinetic energy operator, the second term the nucleus-electron attraction energy operator, and the third term the electron-electron interaction energy operator. This can be more compactly expressed as

$$\hat{H} = \hat{T} + \hat{V}_{ne} + \hat{V}_{ee}$$

Equation 3

with the total energy equal to:

$$E = E_{ke} + E_{ne} + E_{ee}$$

Equation 4

The total energy is a functional of the wave function and can be written as:

$$E[\Psi] = <\Psi|\hat{H}|\Psi> = \int \Psi^* \hat{H} \Psi d\bar{x}$$

Equation 5

where Ψ equals $\Psi(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_N)$ and is normalized. $\bar{x}_i$ represents both the spatial coordinate $\bar{r}_i$ and the spin coordinate s_i.

For an N-electron system, the ground state energy, E_0, and the ground state wave function, Ψ_0, can be determined using the variational principle, *i.e.*,

$$E_0 = \min_{\Psi} E[\Psi],$$

Equation 6

where "$\min_{\Psi}$" signifies minimization with respect to all possible N-electron wave functions. (The notation is from Reference 1 and references within).

Hohenberg and Kohn derived equivalent equations, but use the electron density instead, which can be defined in terms of the density matrix[2] $\gamma(\bar{r}_1, \bar{r}_1')$,

$$\gamma(\bar{r}_1, \bar{r}_1') = N \sum_s \int \Psi^*(\bar{r}_1 s, \bar{x}_2, \ldots, \bar{x}_N) \Psi(\bar{r}_1' s, \bar{x}_2, \ldots, \bar{x}_N) d\bar{x}_2 \ldots d\bar{x}_N$$

Equation 7

The diagonal matrix element of the density matrix (where $\bar{r} = \bar{r}_1 = \bar{r}_2$) is the density $\rho(\bar{r})$. Through this definition, the wave function maps to the electron density. The total electron density should be greater than or equal to zero. Integration of the electron density over all space should yield the total number of electrons, N, *i.e.*,

$$\int \rho(\bar{r}) d\bar{r} = N$$

Equation 8

According to the first theorem of Hohenberg and Kohn,[5] **Equation 4** can be rewritten in terms of a unique electron density:

$$E[\rho] = E_{ne}[\rho] + E_{ke}[\rho] + E_{ee}[\rho].$$

Equation 9

In other words, there exists a one-to-one relationship between the electron density of the system and its total energy. The external potential due to the nuclei can then be expressed as:

$$E_{ne}[\rho] = \int \rho(\bar{r})\, \upsilon(\bar{r})\, d\bar{r}$$

Equation 10

and the other two terms can be combined into a single functional:

$$F_{HK} = E_{ke}[\rho] + E_{ee}[\rho] = E_{ke}[\rho] + J[\rho] + E_{xc}[\rho].$$

Equation 11

$J[\rho]$ represents the classical coulombic repulsion energy,

$$J[\rho] = \tfrac{1}{2}\iint \frac{\rho(\bar{r}_1)\rho(\bar{r}_2)}{|\bar{r}_2 - \bar{r}_1|} dr_1 dr_2$$

Equation 12

$E_{xc}[\rho]$ is a nonclassical term that includes the quantum mechanical electron exchange-correlation energy. Both this and the kinetic energy functional will be discussed in the next section.

The second theorem of Hohenberg and Kohn supports the use of the variational principle with **Equation 9** to find the minimum ground state energy with respect to the electron density, *i.e.*,

$$E_0 = \min_{\rho} E[\rho] = \min_{\rho} F_{HK}[\rho] + \int \upsilon(\bar{r})\, \rho(\bar{r}) d\bar{r}$$

Equation 13

This is equivalent to saying that $\rho(\bar{r})$ is a solution of the stationary principle which can be written:

$$\delta(E[\rho] - \mu[\int \rho(\bar{r})d\bar{r} - N]) = 0$$

Equation 14

where δ represents taking the differential of the functional and μ is the chemical potential,

$$\mu = \frac{\delta E[\rho]}{\delta\rho(\bar{r})} = \upsilon(\bar{r}) + \frac{\delta F_{HK}[\rho]}{\delta\rho(\bar{r})}$$

Equation 15

Thus, the ground state energy can be determined via the ground state electron density if F_{HK} is known.

1.2.3. Kohn and Sham Equations

Hohenberg and Kohn did not state how to calculate the ground state density without knowledge of the wave function, nor did they give an explicit form for F_{HK}. Kohn and Sham determined a useful method that goes beyond simple approximations.

Kohn and Sham invented an indirect approach to accurately calculate the kinetic energy. They introduced a reference system of non-interacting electrons whose electron density is exactly the same as that of the system of interest. This allows the many-body problem to be reduced to N independent particle problems.

The total energy of a system in an external field $\upsilon(\bar{r})$ can be written as:

$$E[\rho] = \int \rho(\bar{r})\upsilon(\bar{r})d\bar{r} + J[\rho] + E_{ke}^{ref}[\rho] + E_{xc}[\rho],$$

Equation 16

where J[ρ] is defined in **Equation 12**. E_{ke}^{ref} is the kinetic energy of the reference system, and

$$E_{xc}[\rho] = E_{ke}^{real}[\rho] - E_{ke}^{ref}[\rho] + E_{ee}^{total}[\rho] - J[\rho]$$

Equation 17

The exchange-correlation term has a kinetic energy component. This gives a correction to the kinetic energy due to the approximate nature of the reference system. Also, because $E_{ee}^{total}[\rho]$ describes the total electron interaction energy, the classical portion must be subtracted, leaving only the quantum mechanical effects.

The minimization of Equation (16) yields:

$$\frac{\delta E[\rho]}{\delta\rho(\bar{r})} = \upsilon(\bar{r}) + \frac{\delta J[\rho]}{\delta\rho(\bar{r})} + \frac{\delta E_{ke}^{ref}}{\delta\rho(\bar{r})} + \frac{\delta E_{xc}}{\delta\rho(\bar{r})},$$

Equation 18

Or in equivalent terms:

$$\mu = \upsilon(\bar{r}) + \int \frac{\rho(\bar{r}_2)}{(\bar{r}_1 - \bar{r}_2)} dr_2 + \frac{\delta E_{ke}^{ref}}{\delta\rho(\bar{r})} + \upsilon_{xc}(\bar{r}).$$

Equation 19

The second term is derived from **Equation 12** and υ_{xc} is defined by:

$$\upsilon_{xc}(\bar{r}) = \frac{\delta E_{xc}(\rho)}{\delta\rho(\bar{r})}$$

Equation 20

By combining the first, second and last terms in **Equation 19**, the chemical potential becomes:

$$\mu = \upsilon_{eff} + \frac{\delta E_{ke}^{ref}(\rho)}{\delta\rho(\bar{r})}$$

Equation 21

The electron density can be obtained by solving the N one-electron equations:

$$[-\tfrac{1}{2}\sum_i \bar{\nabla}^2 + \upsilon_{eff}(\bar{r})]\Psi_i = \varepsilon_i \Psi_i,$$

Equation 22

and setting

$$\rho(\bar{r}) = \sum_{i}^{N} \sum_{s} |\Psi_i(\bar{r},s)|^2$$

Equation 23

Equation 20, **Equation 22** and **Equation 23** are called the Kohn-Sham (KS) equations. **Equation 23** must be solved self-consistently due to the dependence of υ_{eff} on $\rho(\bar{r})$. This is analogous to solving for Ψ and E in the Hartree approximation,[10] the main difference being that the exact HF exchange term is replaced by the E_{xc} term. One begins with a guess for the electron density to build υ_{eff}. This is plugged into **Equation 21** which gives a new $\rho(\bar{r})$. Then the new $\rho(\bar{r})$ is used to construct a new υ_{eff}. Once the convergence criteria are met, the total energy is given by:

$$E = \sum_{i}^{N} \varepsilon_i - \tfrac{1}{2}\int \frac{\rho(\bar{r}_1)\rho(\bar{r}_2)}{|\bar{r}_2 - \bar{r}_1|} + E_{xc}[\rho] - \int \upsilon_{xc}(\bar{r})\rho(\bar{r})d\bar{r}$$

Equation 24

This equation gives in principle the exact ground state energy. However, the exact form of $E_{xc}[\rho]$ is still unknown. Instead models of the exchange-correlation functional are used to approximate the exchange-correlation energy. These will be described next.

1.2.4. Exchange-correlation Functional

1.2.4.1. Local Density Approximation

To solve the KS equations, an explicit form for E_{xc} is still needed. The simplest functional is modeled after the uniform electron gas and is known as the local-density approximation (LDA). In general, the functional is written as:[1]

$$E_{xc}^{LDA}[\rho] = \int \rho(\bar{r})\varepsilon_{xc}(\rho)d\bar{r}\,.$$

Equation 25

$\varepsilon_{xc}(\rho)$ indicates the exchange and correlation energy per particle of a uniform electron gas of density ρ. The corresponding form for υ_{xc} is:

$$\upsilon_{xc}^{LDA}[\rho] = \frac{\delta E_{xc}^{LDA}}{\delta\rho(\bar{r})} = \varepsilon_{xc}(\rho(\bar{r})) + \rho(\bar{r})\frac{\partial\varepsilon_{xc}(\rho)}{\partial\rho}\,.$$

Equation 26

$\varepsilon_{xc}(\rho)$ can be divided into the exchange and the correlation components:

$$\varepsilon_{xc}(\rho) = \varepsilon_x(\rho) + \varepsilon_c(\rho)\,,$$

Equation 27

where

$$\varepsilon_x(\rho) = -\tfrac{3}{4}(\tfrac{3}{\pi})^{\frac{1}{3}}\rho(\bar{r})^{\frac{1}{3}}$$

Equation 28

Vosko *et al.*[11, 12] have constructed an accurate form of $\varepsilon_c(\rho)$ in the local density approximation:

$$\varepsilon_c^{VWN}(\bar{r}) = \tfrac{A}{2}\{\ln\tfrac{x}{X(x)} + \tfrac{2b}{Q}\tan^{-1}\tfrac{Q}{2x+b} - \tfrac{bx_0}{X(x_0)}[\ln\tfrac{(x-x_0)^2}{X(x)} + \tfrac{2(b+2x_0)}{Q}\tan^{-1}\tfrac{Q}{2x+b}]\}$$

Equation 29

where $x=r^{1/2}$, $X(x)=x^2+bx+c$, and $Q=(4c-b^2)^{1/2}$. For a system with paired spins, which applies to most of the systems of interest in this thesis, the values for the constants are: A=0.0621814, x_0=0.4009286, b=13.0720, and c=42.7198. For strongly bound systems LDA gives good molecular structures, vibrational frequencies and charge densities, but gives poor thermochemical properties.[3]

1.2.4.2. Generalized Gradient Approximation

Atoms and molecules are not homogenous in their electron densities. Therefore, to improve on the description used by the LDA, a density gradient dependence was incorporated into E_{xc}. The general form of the exchange-correlation functional is:[3]

$$E_{xc}^{GGA} = \int f(\rho(\bar{r}), |\nabla\rho(\bar{r})|)d\bar{r}$$

Equation 30

where $f(\rho(\bar{r}), |\nabla\rho(\bar{r})|)$ is a "suitably chosen" function of its two variables. Like the LDA method, the E_{xc} is usually split into two parts. An example of a GGA exchange functional is one by Becke:[13]

$$E_x^{B88}[\rho] = -\tfrac{2}{3}(\tfrac{3}{\pi})^{\frac{1}{3}}\int \rho(\bar{r})^{\frac{4}{3}} d\bar{r} - \int \rho(\bar{r})^{\frac{4}{3}} \frac{bx^2}{1 + 6bx\sinh^{-1}(x)} d\bar{r}$$

$$x = \rho^{-\frac{4}{3}} |\nabla\rho| \qquad\qquad b=0.0042$$

Equation 31

The first term reproduces the exchange energy of a uniform electron gas and the second term introduces a correction through the gradient $\nabla\rho$. Becke found that the gradient correction to exchange was more significant than that to correlation.[14, 15]

A popular GGA correlation functional is one by Lee, Yang and Parr:[16]

$$E_c^{LYP}(\bar{r}) = -a\int \frac{1}{1 + d\rho(\bar{r})^{-\frac{1}{3}}}\{\rho(\bar{r}) + b\rho(\bar{r})^{-\frac{2}{3}}[C_F\rho(\bar{r})^{\frac{5}{3}} - 2t_w + (\tfrac{1}{9}t_w + \tfrac{1}{18}\nabla^2\rho(\bar{r}))]e^{-c\rho(\bar{r})^{-\frac{1}{3}}}\}d\bar{r}\,.$$

Equation 32

The constants have the following values: a=0.04918, b=0.132, c=0.2533, and d=0.349. t_W is a local "Weizacker" kinetic-energy density and is expressed as:[16]

$$t_w(\bar{r}) = \tfrac{1}{8}\frac{|\nabla\rho(\bar{r})|^2}{\rho(\bar{r})} - \tfrac{1}{8}\nabla^2\rho(\bar{r})\,.$$

Equation 33

1.2.4.3. Hybrid Functionals

Although GGA functionals have been successful at including the deficiencies of the LDA approach, they still only estimate the exchange energy and non-local effects. Becke has devised a mixed HF-DFT functional called Becke3LYP in order to add these essential components. This functional has been shown to perform significantly better than previously developed functionals.[17] The key is the addition of the rigorously exact and non-local Hartree-Fock exchange. The Becke3LYP functional can be written as:[17]

$$E_{xc}^{B3LYP} = E_{xc}^{LDA} + a_0(E_x^{HF} - E_x^{LDA}) + a_x\Delta E_x^{B88} + a_c\Delta E_c^{PW91}$$

Equation 34

where a_0=0.20, a_x=0.72, and a_c=0.8. These are semiempirical coefficients determined from fitting 56 atomization energies, 42 ionization potentials, 8 proton affinities and 10 first-row total atomic energies.[17] Obviously, these fitted parameters also increase the accuracy of the functional. The LDA exchange term was defined above, but the correlation functional is by Perdew and Wang.[18] The ΔE terms are gradient corrections to the Becke88 exchange functional (defined above) and the Perdew-Wang 1991 correlation functional.[19]

1.3. Pseudopotentials

Electronic structure problems are typically solved using the Schrödinger equation. This equation makes an assumption that particles move at speeds far below the speed of light. That is true for electrons in light elements, however, heavy elements in the fifth row of the periodic table and below present a problem. In these atoms the core electrons move at speeds comparable to the speed of light, so that the Schrödinger equation breaks down. For example, it predicts atomic radii to be too large. The properties of such atoms can be successfully predicted using the Dirac equation. Unfortunately, this equation is a lot harder to solve than the Schrödinger equation, which makes it impractical for molecular systems.

Pseudopotentials allow the use of Schrödinger equation for heavier elements like rhodium and molybdenum. In this formalism the core electrons are replaced by a relativistic pseudopotential that shields the valence electrons from most of the nuclear attraction and also makes the wave function orthogonal to all the core states. The reason this approach works is that relativistic pseudopotentials allow for the contraction of electrons.

An additional benefit of core pseudopotentials is that they reduce the number of electrons described by the Schrödinger equation. Without the core electrons, the calculations require less processor time and memory.

1.4. Methods used in modeling surfaces

In simulating surfaces using electronic structure theory researchers frequently use different methodology than when they are simulating isolated molecules. One way to simulate a surface is to use a cluster model. This approach has both advantages and disadvantages. On the one hand, one can use high-level electronic structure calculations. On the other hand, there are edge effects that may introduce significant errors.

Another way to model a surface is by means of a slab model. In this approach one simulates a periodically repeating system that has a structure close to that of bulk material, except it is broken into slices with layers of vacuum in between. Unlike the cluster-based surfaces models, the slab-based surface models are not plagued by edge effects; however, slab models do not allow the efficient use of high-level quantum calculations.

Although one can use conventional Gaussian-based basis sets with slab models, today plane-wave basis sets are more popular for dealing with periodic systems. Unlike the Gaussian-based basis sets plane-wave basis sets are not centered on atoms.

Plane-wave basis sets have both advantages and disadvantages compared with Gaussian-based basis sets. Plane-wave basis sets treat all points in space equally. On the one hand, this is free of basis set superposition errors. On the other hand, this means that the empty space between the slabs is described as well as the space inside and around the atoms, which is a waste of computational resources. To achieve the same accuracy as with the Gaussian-based basis sets, one needs more plane-wave functions than Gaussian-based functions. However, because plane waves are simpler in form than Gaussian-based basis functions, each operation involving plane waves is faster. Plane-wave basis sets compensate for the large number of basis set components by the speed of operations on each of these components.

Describing nuclear cusps in the electronic wave function with a plane wave basis set requires a very large number of plane waves; however, these regions contribute little to the chemical properties. That is why effective core pseudopotentials (ECPs) are used more in plane-wave calculations than they are in calculations that use Gaussian-based basis sets. ECPs used in plane-wave calculations not only account for relativistic effects associated with core electrons and reduce the number of electrons that have to be treated, but also remove the nuclear cusps in the electronic wave function, making it possible to use a smaller number of plane waves.

1.5. Changes in Computational Methods

As the research described in this dissertation was being done, we gradually switched to more and more sophisticated methods. This is particularly evident in chapters 3, 4 and 5. In Chapter 3 we modeled reactions by studying the minima and relying on their relative electronic energies, found using density functional theory with the PW91 functional.[19] In Chapter 4 we switched to the B3LYP functional[20] and instead of only considering the minima we also started to consider the electronic energies of transition states. Our methods have gradually improved. Later we started to consider thermochemistry. We not only considered the electronic energies of minima and transition states but also their Gibbs free energies. Finally, in Chapter 5 we have also included the interactions of intermediates and transition states with a polar solvent using the COSMO solvation model.[21]

2. DIMENSIONAL STRATEGIES FOR GLOBAL OPTIMIZATION

2.1. INTRODUCTION

2.1.1. Overview of the global optimization problem

Minimization problems are frequently encountered in science and engineering. Unfortunately, such problems are hard to solve for systems with a large number of degrees of freedom. Often the number of minima grows at least exponentially with the number of degrees of freedom[22,23]. Most non-stochastic optimization algorithms can easily find one minimum, but usually it is not the global minimum[24].

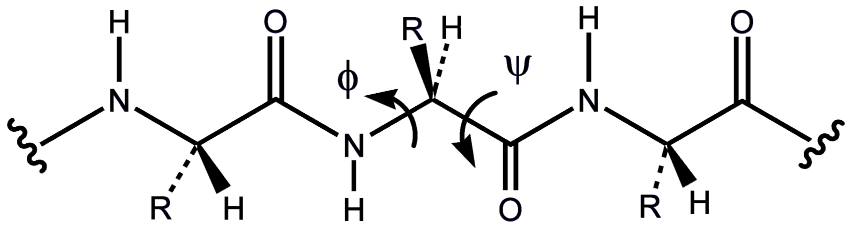


Figure 1 An illustration of protein structure. Only three amino acid monomers are shown.

One very important global optimization problem is protein folding[25]. A protein is a polymer made up of amino acids. (See **Figure 1**) Most amino acids have two easily rotatable bonds in the backbone, and many also have flexible side chains that add additional degrees of freedom to the protein. It is not computationally feasible to exhaustively sample all of a protein's configuration space, even if one considers only the torsional degrees of freedom.

2.1.2. Genetic algorithms

Genetic algorithms have been successfully applied to global optimization problems[26,27,28,29]. These algorithms essentially simulate evolution within a population, where each member of population is an alternative solution to the problem, for example a molecular geometry. The data that makes up the solution is treated as genetic information. Initially many random solutions are introduced. Those with lower energy (for problems for which it is the energy that is being minimized) are allowed to breed and those with higher energy are discarded. Breeding is done by making new solutions consisting of the parent genetic material, making a few mutations. Genetic algorithms have several advantages. They do not require gradients or continuous coordinates, and they perform well on rough surfaces. However genetic algorithms have trouble finding global minima, and require a large number of function evaluations.[30, 31, 32] Another downside of genetic algorithms is their poor performance when dealing with problems that have a large number of degrees of freedom.[33]

2.1.3. Monte-Carlo-based methods

There are several Monte-Carlo[34] based methods for global optimization. One of the earliest methods to be successfully applied to a range of global optimization problems is simulated annealing[35]. In this method one runs a constant temperature Monte-Carlo simulation, gradually lowering the temperature of the simulation in the hope that in the zero temperature limit the system will converge to its global minimum. In practice, however, this method runs into trouble when the entrance to the "basin" leading to the global minimum is narrow.

A closely related Monte-Carlo approach is parallel tempering[36,37,38]. It relies on conducting Monte-Carlo walks in parallel at different temperatures $T_1 < T_2 < T_3 <\ldots< T_n$. Each simulation is referred to as replica and the algorithm allows for occasional exchanges of configurations from different replicas. In this approach the highest temperature is chosen so that potential energy barriers are readily overcome. Exchanging configurations between replicas provides a means for the lower temperature replicas to "get around" barriers.

2.1.4. Quantum annealing

Another global minimization method is quantum annealing[39,40]. Instead of treating the simulated particles classically, this approach uses quantum mechanical models. Quantum mechanical particles can tunnel through potential energy barriers to access deeper minima. This allows one to reach the global minimum. The system can be brought back to the classical limit by either increasing the masses of the particles or decreasing the value of $\hbar$. Unfortunately these methods do not scale well with increasing system size.

2.1.5. Global optimization methods that modify the potential energy surface

Some optimization methods modify the potential energy surface to make it smoother. For example, Stillinger and Stillinger[41] changed the long-range interactions between atoms. This approach is of limited utility, because the global minimum in the modified potential need not correspond to that in the true potential.

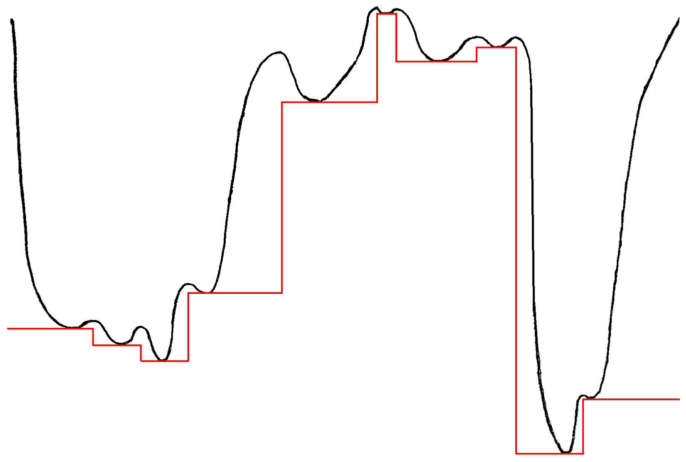


Figure 2 Potential energy landscape used in basin hopping.

It is easier to find the global minimum on the staircase-like potential energy surface used in basin hopping (red) than on the true potential energy surface (black).

Basin hopping[42] is another algorithm that changes the potential energy surface. It performs a Monte-Carlo walk, but instead of using the energy of the current atomic configuration at each step, it uses the energy of the inherent structure which is the potential energy minimum to which the current structure can be optimized. The algorithm transforms the hill-like landscape into a staircase-like landscape, keeping the correct energy and geometry for the global minimum (**Figure 2**).

2.1.6. Artificially increasing the dimensionality of the system

Another approach to optimization involves artificially increasing the dimensionality of the system. It has been known for some time that the maximum number of minima of Lennard-Jones clusters occurs in three dimensions.[47,43,44] **Figure 3** plots the number of minima of LJ_{13} as a function of the number of dimensions. As the number of dimensions is increased from three, the system gradually looses its complexity, until it has only one minimum in ten-dimensional space.

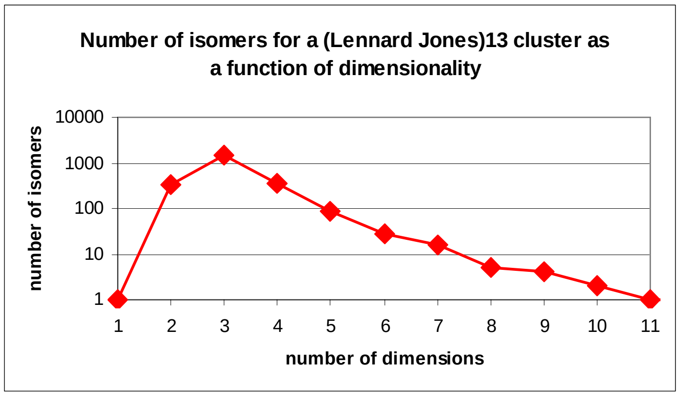


Figure 3 Number of known isomers for a (Lennard Jones)$_{13}$ cluster as a function of dimensionality.

LJ_{13} has the greatest number of isomers in 3 dimensions.

2.1.7. The algorithm of Purisima and Scheraga

Purisima and Scheraga[45, 46] introduced an algorithm in which the global minimum is located in extended dimensionality space, and then the system is "compressed" back to the physical three-dimensional "world". Unfortunately this approach does not work well because compression of the global minimum in an extended dimensional space back to three dimensions is likely to lead to a minimum other than the global minimum (**Figure 4**).

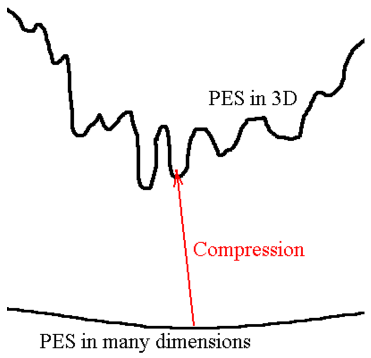


Figure 4 An illustration of the algorithm of Purisima and Scheraga.

Compression of many-dimensional global minimum to 3D puts the system into a low-energy 3D minimum.

2.1.8. The algorithm of Faken *et al.*

2.1.8.1. Overview of the algorithm of Faken *et al.*

Faken *et al.*[47] took a different approach to the problem. Their algorithm is a compromise between basin hopping and dimensional compression. Like basin hopping, their algorithm eliminates potential energy barriers between adjacent minima. However, instead of avoiding them by substituting the inherent structure energy for the energy of current configuration, it exploits extended dimensionality to travel around potential energy barriers to other 3D minima. The important difference between this algorithm and that suggested by Purisima and Scheraga[45,

[46] is that it never descends to minima in the extended dimensional space, and if run long enough, the system arrives at the global minimum.

2.1.8.2. Steps of the algorithm of Faken *et al.*

Here is how the algorithm of Faken *et al*. works. Suppose one wants to find the global minimum of a system described by a set of continuous coordinates R in D dimensions, where normally D = 3.

1. Minimize the potential energy V(**R**) in D dimensions using a non-stochastic algorithm such as steepest descent or conjugate gradient. The system arrives at a local minimum where V(**R**)=E_{lm}.
2. Carry out a random walk at constant energy in D+H dimensions, where H ≥ 1. At the beginning of the walk, the system is in D dimensions, and the extra coordinates are set to 0. At the end of the walk, the energy of the system is still equal to E_{lm}, but the system is someplace in the D+H dimensional space. In the algorithm of Faken *et al.* all atoms are moved simultaneously.
3. Compress the system back to the D dimensional space, keeping the energy constant.
4. Optimize the geometry in D dimensions, setting extra coordinates equal to zero. The energy of the resulting minimum will be equal to or lower than that of the previous minimum (in 3D).
5. The procedure is either repeated by returning to step 2, or the current **R** is reported as the final answer.

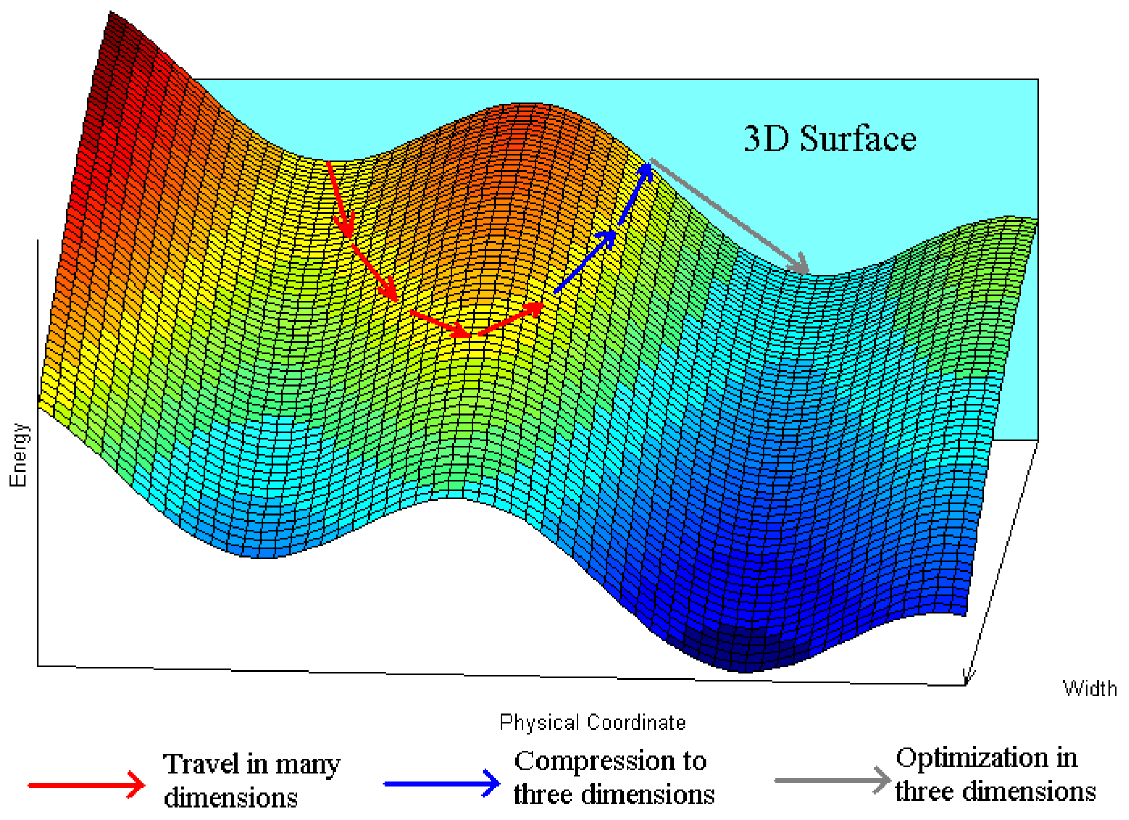

3D Surface
Energy
Width
Physical Coordinate
Travel in many dimensions
Compression to three dimensions
Optimization in three dimensions

Figure 5 Key steps in the algorithm of Faken et al.

2.1.8.3. Compression from 4D to 3D in the algorithm of Faken *et al.*

Compressing the system from D+H to D dimensions at constant energy requires a function that describes how far the system is from the D-dimensional world. This function should be continuous, differentiable, equal to zero when the system is in D dimensions, and positive when the system is in more than D dimensions. Faken *et al*. introduced the "width" function.

$$W(\mathbf{x}) = \sum_{i=1}^{N} \sum_{n=D+1}^{D+H} x_{i,n}^{2} \quad ,$$

Equation 35

where N is the number of atoms in the system and $x_{i,n}$ is the coordinate of the i^{th} atom in the n^{th} dimension. In principle, they could have chosen another functional form for W, for example, $W(\mathbf{x}) = \sum_{i=1}^{N} \sum_{n=D+1}^{D+H} x_{i,n}^4$ or $W(\mathbf{x}) = -H + \sum_{i=1}^{N} \sum_{n=D+1}^{D+H} \cosh(x_{i,n})$.

Faken *et al*. compressed the system from 4D to 3D at constant energy by moving atoms perpendicular to the energy gradient in the direction of decreasing width. In our work we used the same width function as Faken *et al*., but we tried several approaches to compression in addition to that of these authors.

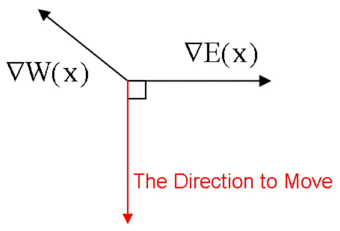


Figure 6 One way to compress the system to 3D is to move perpendicular to the energy gradient in the direction of decreasing width.

2.1.8.4. Advantages and disadvantages of the algorithm of Faken *et al.*

This algorithm has both advantages and disadvantages.

The advantages are:

1. Unlike the algorithm of Purisima and Scheraga[45, 46], if run long enough, it is guaranteed to find the global minimum.
2. The algorithm avoids barriers and high energy local minima.
3. The longer the algorithm runs, the simpler the effective potential energy surface which is accessible.

The disadvantages are:

1. The compression step can encounter local minima of the width function in extended dimensional space, which can slow the algorithm down dramatically.
2. Since the method requires gradients, it will not be straightforward to implement for discrete problems like optimization of circuit board assembly[48] or a regression tree-based QSAR engine[49].

2.1.9. Objectives of our work

The objectives of our work are twofold:

1. to determine if there are advantages to walking in greater than 4 dimensions (Faken *et al*. considered only 4D extended dimensional space.)
2. to develop an efficient method for compression to 3D.

The first step towards reaching our objectives involved implementing the algorithm of Faken *et al*. and applying it to inert gas clusters modeled by the Lennard-Jones potential:

$$E = 4\varepsilon \sum_{i<j} \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right]$$

Equation 36

2.2. DISCUSSION AND RESULTS

2.2.1. Our first implementation of the algorithm of Faken *et al.*

We started by writing a C++ program that implements the algorithm of Faken *et al.*[47] and tested it on 16, 17, 21 and 22 atom Lennard Jones clusters. In our implementation we made a minor change in the optimization algorithm of Faken *et al*.[47] Namely, in cases that the compression step failed to bring the system to 3D, the system was returned to the previous local minimum in 3D rather to the point in 4D from which the compression was started. Our test revealed that this change generally resulted in a small increase in efficiency in locating the global minimum in 3D.

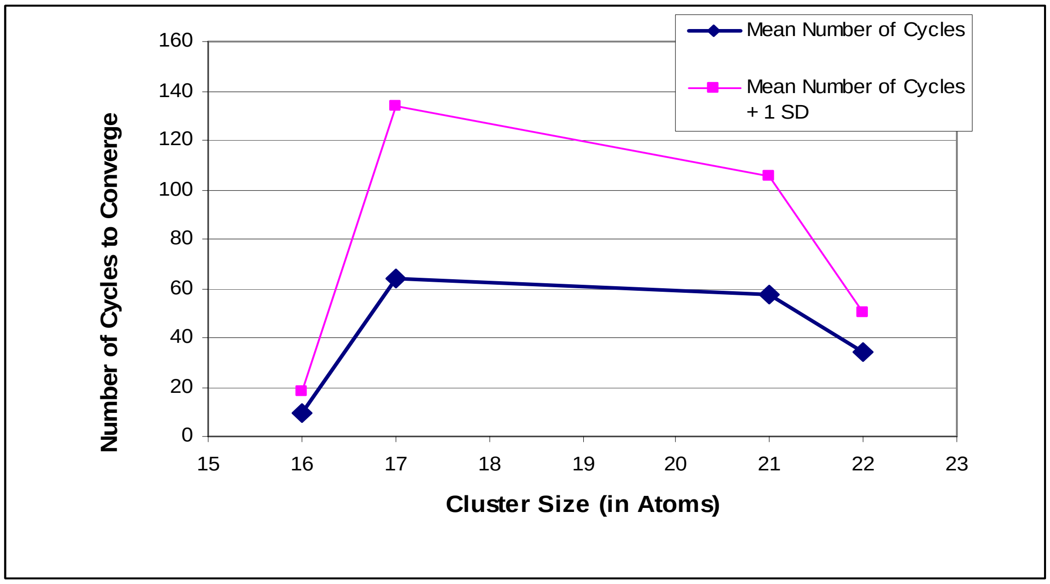


Figure 7 The performance of our first implementation of the algorithm of Faken *et al.*

In terms of the number of cycles required to find global minima, the performance of our code was similar to that of Faken *et al*.[47] The number of cycles remains small even for the largest cluster considered as shown in **Figure 7**. We also found that most of the CPU time was spent in the compression step.

2.2.2. Breaking down the improvement objective into two parts

Based on these results, we formulated our short-term goals. The first objective was to reduce the CPU time per cycle, focusing on the compression step. The second objective was to determine whether the number of cycles could be reduced by "improving" the travel in the 4D step. This is related to the question of what are the optimal walks into 4 (or higher) dimensions in the context of the "ease" of locating the global minimum in 3D space?

2.2.3. Two ways to speed up the compression step

There are two possible approaches, designated "*A*" and "*B*" for future reference, to speeding up the compression step. *A* involves improving the compression step and *B* eliminates it altogether.

2.2.4. Speeding up the compression step by eliminating it altogether

2.2.4.1. A 4D walk that comes back to 3D without compression

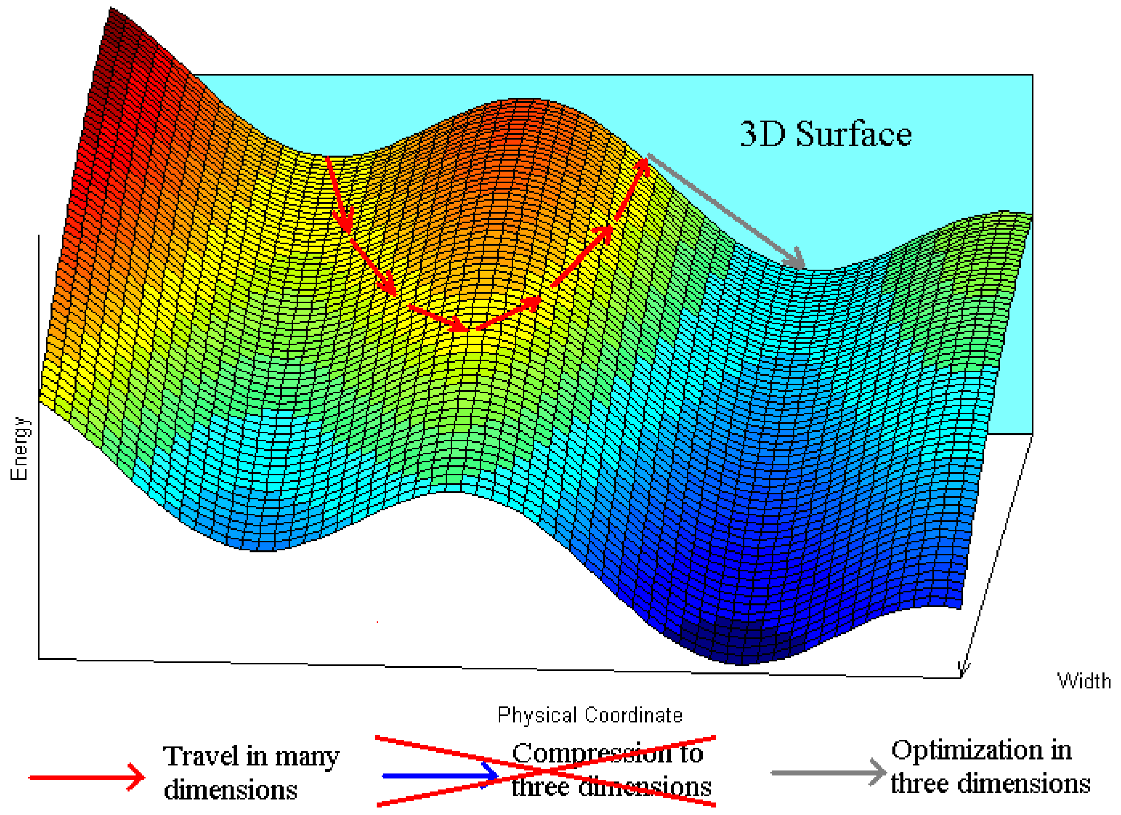

3D Surface
Energy
Width
Physical Coordinate
Travel in many dimensions
~~Compression to three dimensions~~
Optimization in three dimensions

Figure 8 The algorithm of Faken *et al.* without the compression step.

We tried approach *B* first. When traveling at constant energy out of a 3D local minimum and into 4D, there is a possibility of walking around a width barrier and emerging on the other side of it, back in the 3D world, as shown in **Figure 8**. If this happens, there is no need to perform the time-consuming compression step. To accomplish this, we made changes to the way the walk was done. Instead of undergoing random moves at constant energy, we randomly picked a direction of travel in the three-dimensional world, and used the fourth coordinate to keep the energy constant. Throughout this 3D→4D→3D travel the direction of the 3D components of the displacement vector was held constant.

Unfortunately, there are several problems with this algorithm. The first one is evaporation. If one is dealing with a small cluster, many of its atoms are on the surface. When a random direction of 3D travel is picked, many atoms are "ordered" to move away from the cluster, leading to evaporation. (Recall that the walks have to be long enough to proceed between minima, which increases probability of evaporation.) The evaporation problem can be fixed by changing the direction of travel periodically so that runaway atoms head back into the cluster.

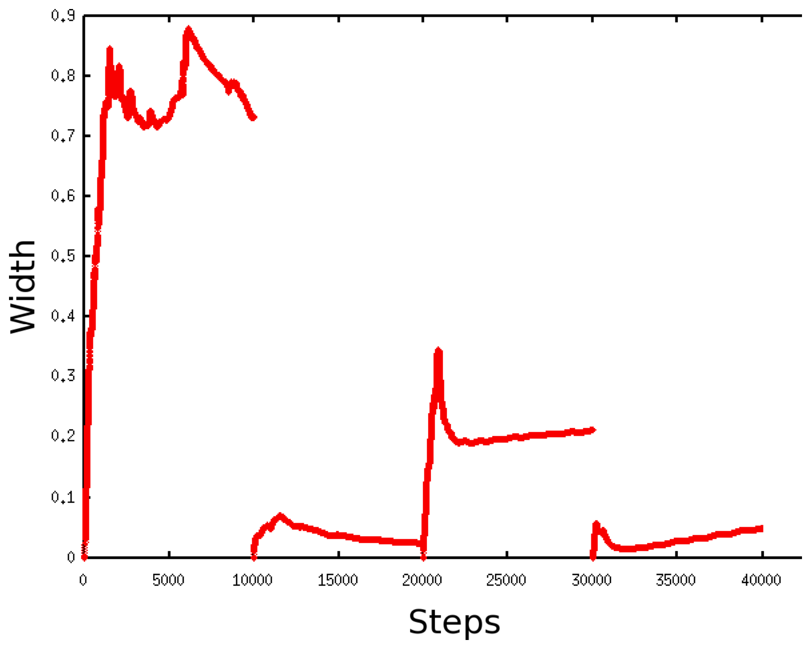


Figure 9 A plot of width for four 10000-step walks at constant energy.

In all four cases at the beginning of the walk the width function is 0, because each walk is started from the 3D "world". Then the width function increases rapidly until it quickly reaches a peak. After that the width functions decreases, but it does not decrease to 0, so the system never returns back to the 3D "world".

The other problem is more serious. There are many non-3D degrees of freedom, which makes the 3D regions only a small part of the 4D space, e.g. for a n atom cluster, moving to the

4D universe introduces n extra degrees of freedom. Thus, although the algorithm goes through many width basins, it rarely hits the small 3D regions in them. This problem is very similar to the problem of the fraction of the volume of a hypersphere inscribed into a hypercube. The greater the number of dimensions, the smaller the ratio of hypersphere/hypercube volume.

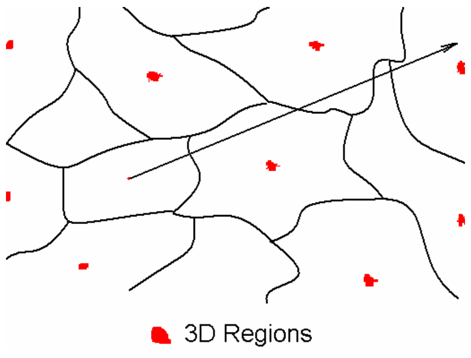


Figure 10 3D regions are so small in hyperspace that hitting them by a simple walk is very improbable.

The above discussion identifies a potential problem with the algorithm Faken *et al.* The motivation behind the approach is that by increasing dimensionality, the complexity of the potential energy surface is decreased. However, to some extent the decrease in complexity of the PES is compensated by having to deal with the complexities of the width function at constant energy. This problem is present for both approaches *A* and *B* for dealing with the compression problem.

2.2.4.2. A walk in 3D that avoids atomic collisions

We also explored a new procedure for optimizing directly in 3D which is related to basin-hopping, but which we thought would be faster. Here are the steps of the algorithm.

1. Using a non-stochastic algorithm, minimize the energy of the system.
2. Pick a random direction **U** for all-atom travel.

3. Take a small all-atom step in the direction **U**.
4. If the walk arrives at a minimum in energy on the line on which it is traveling
    a. Find the energy of the inherent structure associated with the minimum on the line the algorithm is traveling and use it together with the Metropolis rules to determine whether to accept the step.
    b. If the step is accepted, go back to 2 using the new inherent structure. Otherwise go to step 3.
5. If a specified number of steps is exceeded, return to the minimum from which the current walk was started and continue with step 2.

In principle, if run long enough, this algorithm will find the global minimum.

The basin-hopping algorithm[42] requires optimizing to an inherent structure at every step, and frequently the resulting inherent structure is the same as that found in the previous step. In the algorithm described above, optimization to an inherent structure is not carried at each step. Rather, initiation of a search for an inherent structure is undertaken only if a maximum is crossed on the line being followed.

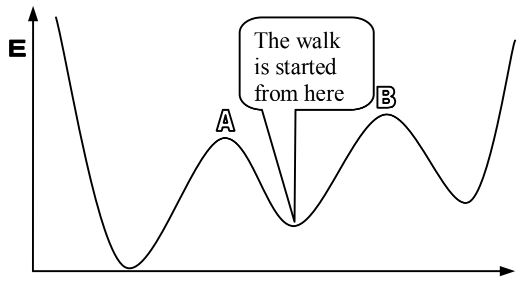


Figure 11 The advantage of the walk described here over basin-hopping.

All configurations between A **and** B **have the same inherent structure. The advantage of the 3D travel algorithm described here over basin-hopping is that it will not spend any time trying to find inherent structures between points** A **and** B**.**

Such an algorithm has been implemented, but there is a serious problem with it. Very frequently two or more atoms come so close together that the system ends up way up on the repulsive wall. This obscures detecting basins. A way to avoid such atomic collisions was found. One can break the direction of travel into an overall direction and a local direction of travel. Before conducting a walk, one selects an overall direction of travel. In each step of the walk, a step is done in a direction that is a sum of the overall direction of travel and a fraction of the direction opposite to the gradient of energy. The size of this fraction is fixed. When atoms are about to collide, the gradient adjusts the direction of motion so that the atoms just brush off each other. This method of walking is illustrated in **Figure 12**. In some respects, it is similar to elastic band methods for finding transition states.[50, 51, 52]

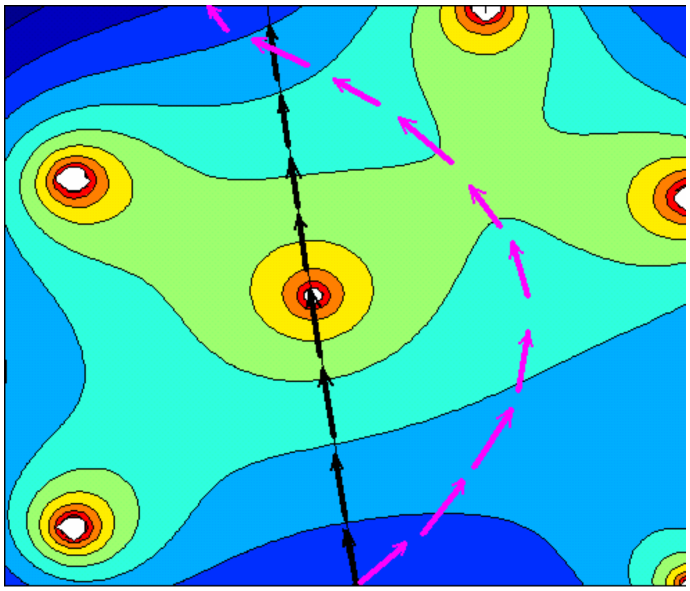


Figure 12 Two alternative ways to travel in 3D

→ Travel in a fixed direction. Unfortunately, it goes through the regions of very high energy.

→ Travel where the direction is corrected at every step to avoid atomic collisions.

This method is very efficient for traveling between different energy minima. An unfortunate thing is that most of the time the minima it finds are not lower in energy than the previous minimum. Although this approach is inefficient for global optimization, it could find uses in searching for transition states. One advantage that it shares with elastic band methods is that it does not require second derivatives. This method also found further use in our work, as will be discussed below.

2.2.5. Improving the algorithm for 4D→3D compression

2.2.5.1. Combining energy and width into a single function

Because eliminating compression to 3D at constant energy did not work, we redirected our efforts to approach *A*, speeding up the computationally costly compression step. One strategy explored involved combining the energy and width into a single function

$$F(\mathbf{x}) = (E(\mathbf{x}) - E_0)^2 + W(\mathbf{x})$$

Equation 37

which describes how far a given point is from the desired point, where $E(\mathbf{x}) = E_0$ and $W(\mathbf{x}) = 0$.

Compressing the system to 3D at constant energy requires finding any root of $F(\mathbf{x})$. One advantage of this approach is that the root of this function is also located at a minimum, so the problem can be solved not only with root-finding, but also with minimization algorithms.

2.2.5.2. Compression using the steepest descent algorithm

The first minimization algorithm we tried for this purpose was steepest descent[53]. This algorithm minimizes the value of $F(\mathbf{x})$ by finding the direction of $-\nabla F(\mathbf{x})$, and moving the system in that direction until a minimum value of $F(\mathbf{x})$ is reached on that line. These steps are repeated until the algorithm comes close to a minimum of $F(\mathbf{x})$.

Although the steepest descent algorithm[53] successfully located local minima of F(**x**), it required a large number of steps. The landscape of F(**x**) is reminiscent of a curved canyon. Optimizing this function by steepest descent is analogous to following the path along the bottom of the canyon. The direction of motion has to be changed frequently to avoid running into walls.

2.2.5.3. Compression using the conjugate gradient algorithm

To reduce the number of steps needed we switched to the conjugate gradient minimization algorithm[53], which resulted in an order of magnitude improvement in the speed of the algorithm.

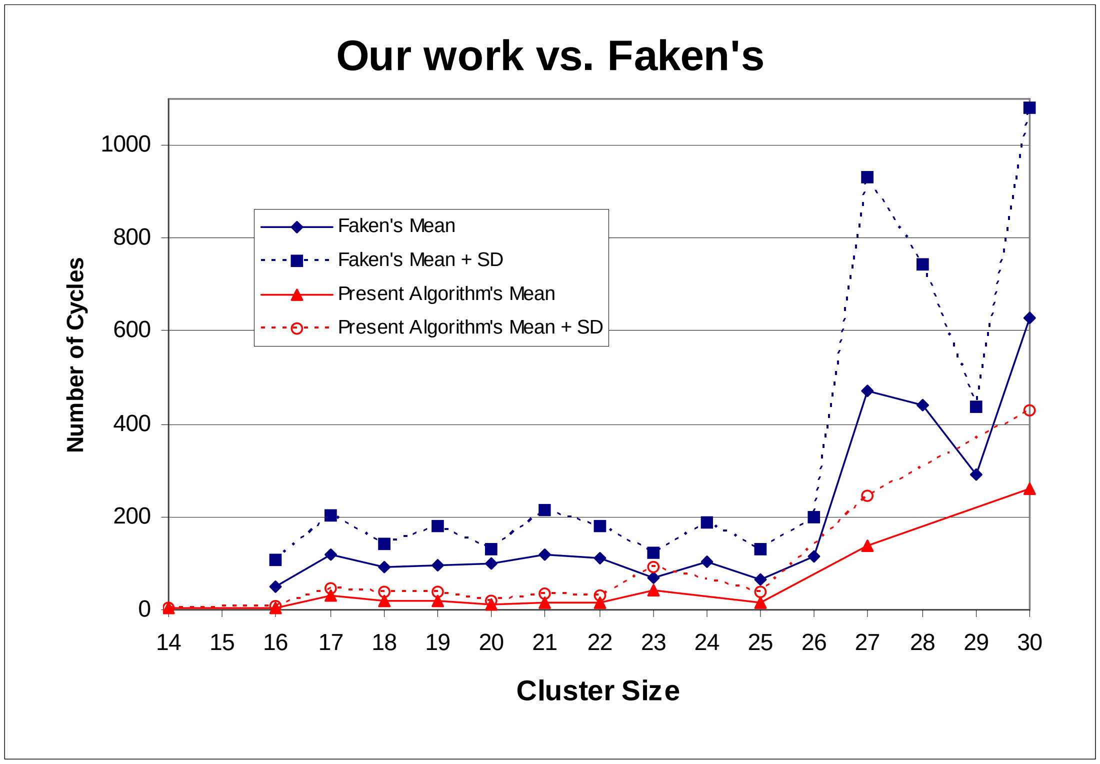


Figure 13 Comparison of performance of the version of our program that uses conjugate gradient algorithm for compression from 4D to 3D with the program of Faken *et al.*

We optimized each Lennard-Jones cluster nine times, each time starting from a different random configuration. We report the mean and the standard deviation in the number of cycles needed to reach the global minimum.

A comparison of the performance of our algorithm which uses conjugate gradient compression of F(**x**) and that of Faken *et al*. is shown in **Figure 13**. For all cases the mean number of cycles required to locate the global minimum was less with our algorithm than with that of Faken *et al*. Both programs show similar trends in performance with increasing cluster size. As the number of atoms increases from 17 to 25, the number of cycles needed to find the global minimum does not rise appreciably, but starting at LJ_{27} the number of cycles needed to find the global minimum starts to go up dramatically.

We have used our program to search for the global minimum of LJ_{38} which is known to be a challenging problem. On the potential the '*fcc-like*' global minimum is separated by a large energy barrier from a family of fivefold local minima. The lowest energy '*fcc-like*' and fivefold minima are very close in energy.[54] The global minimum of this cluster was discovered only in 1995.[55], [56] by Doye *et al*., who used the basin-hopping algorithm. On the average, it took 2000 basin-hopping cycles to find it.

We attempted to find the global minimum of LJ_{38} three times using our implementation of the algorithm. One run did not find the global minimum within 10,000 cycles at which point the job was terminated. The other two runs located it in only 39 and 50 cycles. The paper of Faken *et al*. listed information for only one successful optimization of LJ_{38}, requiring about 8000 cycles. Being able to find the global minimum of this cluster in 50 cycles or less was most encouraging.

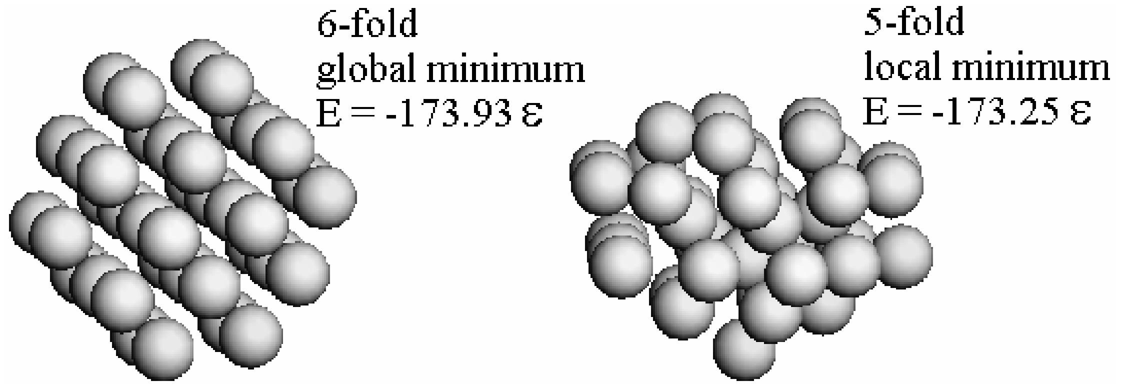


Figure 14 For LJ_{38} the *fcc*-like global minimum is separated from a family of 5-fold local minima by a large energy barrier.

Although, in terms of the number of cycles required, the algorithm that uses conjugate gradient compression has acceptable performance, it is still slow. On average it took 6 min 17 s to optimize LJ_{20} to its global minimum on a computer with a 1 GHz Intel Pentium III processor. Further acceleration of the compression step was needed.

2.2.5.4. Our own compression algorithm

When we optimize F(**x**), the function that incorporates both the width and energy constraint, the system is being taken to a configuration where W(**x**) = 0. This suggests that the method originally developed for avoiding atomic collisions might be useful. In particular, in each step, the system is moved in the direction of

$$\mathbf{D} = \frac{-\nabla F(\mathbf{x})}{\left|-\nabla F(\mathbf{x})\right|} + a\frac{-\nabla W(\mathbf{x})}{\left|-\nabla W(\mathbf{x})\right|}, \quad \mathbf{0 \le a \le 1}$$

Equation 38

until the lowest point on the line is reached. Although -∇F(**x**) already has a width component in it, it is dominated by the energy constraint term. This is the reason that the steepest descent algorithm frequently changes direction and converges slowly. The algorithm described here takes much larger steps. Unlike the steepest decent algorithm, it does not accurately follow the lowest path in the F(**x**) canyon. Instead, it rapidly goes to the 3D surface. Switching the compression routine from conjugate gradient to the algorithm described above with *a* = 0.999 (several values of *a* were tried) led to a dramatic improvement in performance. The mean time needed to find the global minimum of LJ_{20} dropped from 6 min 17 s to 23 s.

2.2.6. Improving travel from a 3D local minimum into 4D space at constant energy

2.2.6.1. What can be achieved by improving the 3D→4D travel at constant energy

While working on improving the algorithm for compressing the system from 4D to 3D at constant energy, we also worked on ways to improve travel from a 3D local minimum into 4D space at constant energy. Although this step took a well under 0.1% of the run time, it was hoped that 3D → 4D travel could be "designed" so as to put the system into places that are more likely to be compressed to low-energy 3D energy minima, thereby requiring fewer cycles to find global minima.

2.2.6.2. The procedure Faken *et al.* used for the 3D → 4D travel step

In each cycle, Faken *et al*. accomplished the 3D → 4D travel step by making 100 all-atom moves for Lennard-Jones clusters with 30 or fewer atoms, and 50 such moves for LJ_{38}. Each move consisted of displacing every atom by a distance of 0.1 σ in the direction, perpendicular to the gradient of energy.

2.2.6.3. Improving the 3D → 4D travel step by increasing the dimensionality

In their paper Faken *et al*. speculated that performance of their algorithm would be improved by using more dimensions for traveling out of 3D local minima, the logic being that the more extra dimensions you add, the easier it is to overcome barriers. We decided to test that hypothesis. We ran the same test on small Lennard-Jones clusters but traveled in 5D and 6D instead of 4D. The results, reported in **Figure 15**, do not reveal a significant advantage to using these higher dimensionalities.

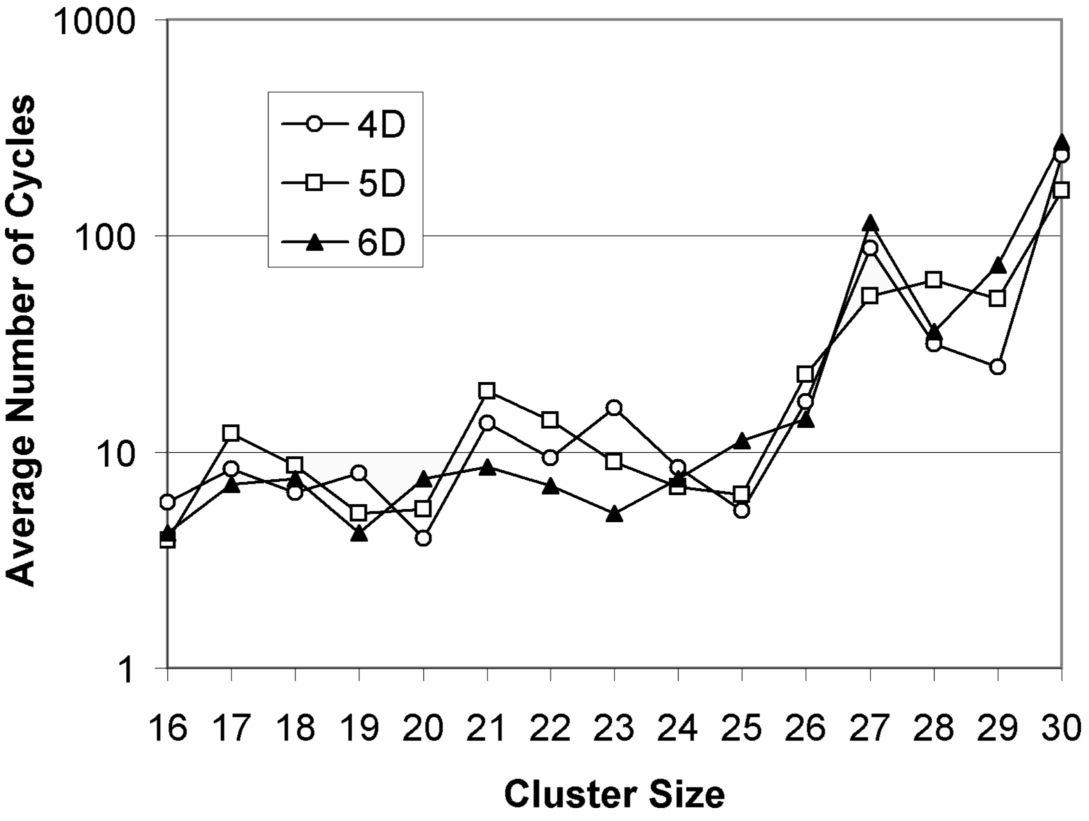


Figure 15 Influence of dimensionality on algorithm performance.

Increasing dimensionality past 4D does not significantly influence the overall performance.

2.2.6.4. Conducting the 3D → 4D travel step by single leap

In the our work described above, the walk out of the 3D local minimum was done by making many small all-atom steps at constant energy, as was done by Faken *et al.* In each step the direction of travel was chosen randomly, under the constraint that it is perpendicular to the energy gradient. This is by no means the only way to travel out of the 3D local minimum. One can instead use big leaps sized according to the central limit theorem. A one-dimensional walk, consisting of N steps of length ΔX each in an independent random direction, results in a probability function for the final position which is a Gaussian distribution centered at the starting position of the walk that has a standard deviation of $\Delta X\sqrt{N}$. Taking a big leap for each atom should give them appropriate displacement distances. In Monte-Carlo simulations this is not acceptable because the energy will most probably rise dramatically, but in our program this is not important. This energy rise can be accounted for by adjusting atomic positions after the big leap and returning the system back to the constant energy surface, that contains the 3D local minimum from which the leap originated.

One motivation for using an algorithm like this is that the algorithm that uses many small steps has three disadvantages.

1) It will always have a Gaussian-like distribution of final positions, while a single-leap can have any distribution of final positions.

2) Our ultimate interest is in the optimization of polymers, for example, proteins. A walk consisting of many small steps will have trouble taking one strand through another. For example, it will have trouble separating two threaded rings, or untying a knot. An algorithm with big leaps could solve this problem.

3) In a non-random walk it is easier to favor special directions, for example, bond rotations.

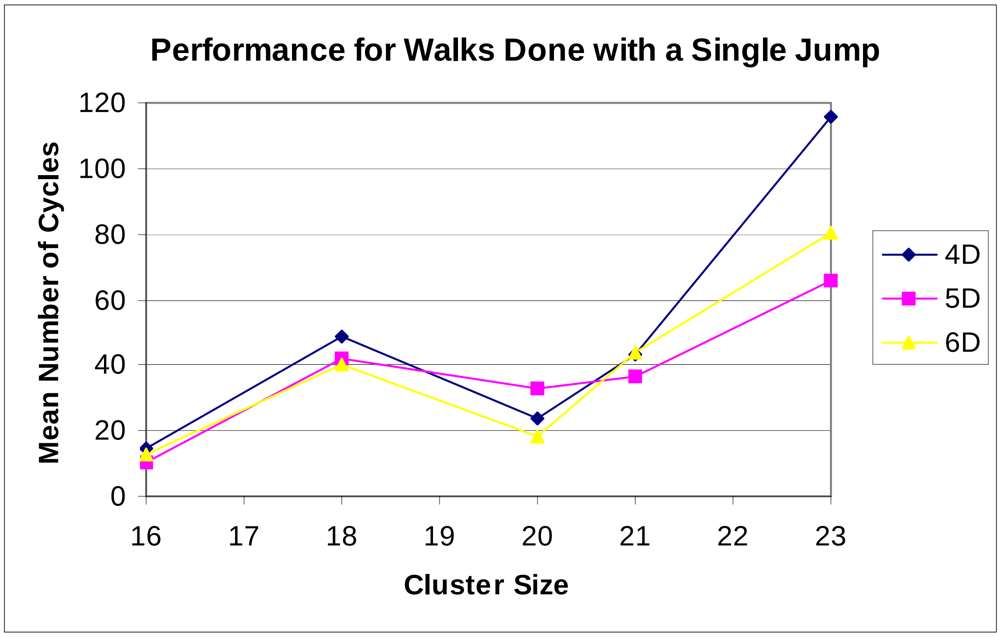


Figure 16 For small LJ clusters doing the walk in a single leap is somewhat less efficient than making many small steps.

We tested this walking technique. The results are shown in **Figure 16**. For small LJ clusters doing the walk in a single leap is somewhat less efficient than making many small steps. As before, the number of dimensions used for walking does not seem to significantly affect the number of cycles needed to find the global minimum.

2.2.6.5. Using the adjustable step size for the 3D → 4D travel step

In their work Faken *et al*. used a fixed step size. For Lennard-Jones clusters Faken *et al*. found reasonable ranges of displacements for 3D→4D, but adjustments will be needed for many other systems. Changing this parameter on the fly would facilitate the use of the algorithm for other problems. For this reason and because it could improve the efficiency we examined use of an algorithm in which the step size is adjusted on the fly so that approximately half of the steps are accepted and half are rejected. If the 3D→4D travel step in the algorithm of Faken *et al*. does not take it far enough from the 3D energy local minimum, the compression to 3D at constant energy step will take the system back to the 3D local minimum and not to a lower energy minimum. On the other hand, if the 3D→4D travel step takes the system too far from the 3D local minimum, the resulting geometry is not likely to be reasonable, and will not be successfully compressed back to 3D at constant energy. We thought, it would be best to keep the 3D→4D travel step between these extremes, by adjusting the range of the displacements at runtime.

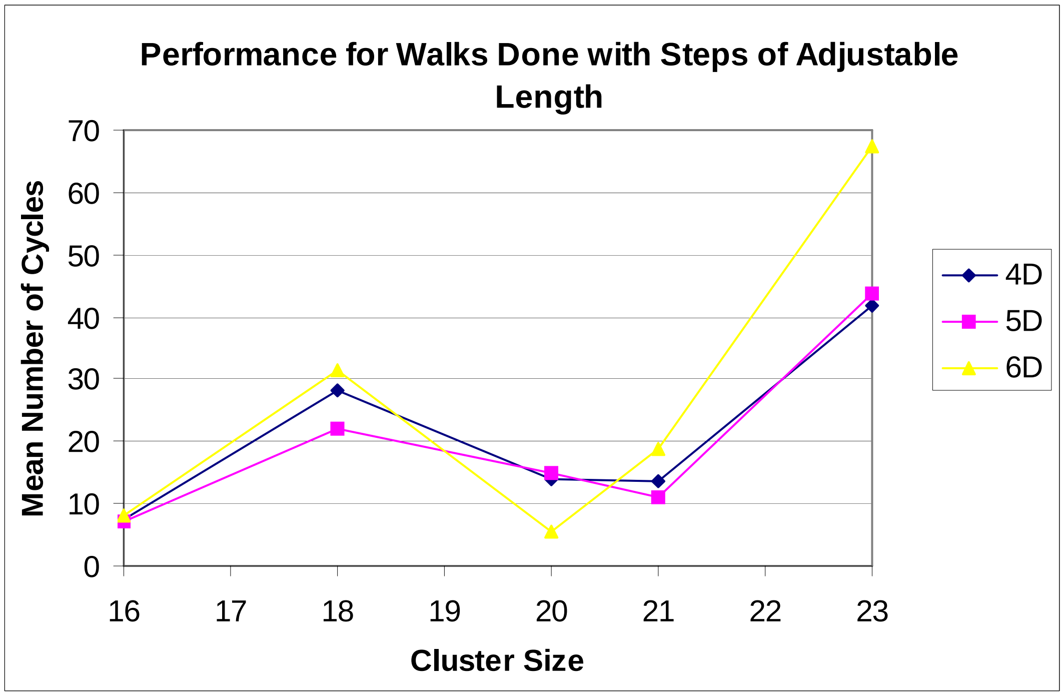


Figure 17 Adjusting the step size during the run does not help with performance.

Again we conducted walks with many small steps, but this time the step size was not fixed. If the algorithm ran into a non-zero local width minimum while compressing to 3D, the step size was decreased. If the algorithm either returned to the original local minimum, or found a better one, the step size was increased. The performance of this algorithm for small Lennard-Jones clusters is shown in **Figure 17** . The performance was about as good as that with constant step size. Again, increasing the number of dimensions used for the walk did not significantly influence the number of cycles needed to converge, because the fourth dimension alone is sufficient to go around any 3D barrier. Although it is true that the potential energy surface for Lennard-Jones clusters becomes simpler when one increases the number of dimensions from 3 to 4 to 5 to 6, the algorithm can’t take full advantage of that, because to stay at constant energy it has to keep the system geometry near the 3D potential energy surface.

2.2.6.6. Changing the magnitude of the extra dimensional components in the 3D → 4D travel step

The role of the three physical dimensions is different from that of extra dimensions, suggesting that the physical and unphysical degrees of freedom could be treated differently. For example, the extra dimensional components of the displacement vector need not have to have the same range as the 3D components. If they are set to 0, and the walking is done by a single leap, the algorithm becomes similar to basin-hopping[42] at 0 K with a somewhat unusual way of finding the geometry of the inherent structure. Instead of finding it by a single non-stochastic geometry optimization, the algorithm at first lowers the energy to the energy of the previous local minimum, and only then proceeds to the energy of the inherent structure.

Both algorithms find the system at high energy after the jump. Basin hopping solves this problem by conducting a non-stochastic geometry optimization in 3D to find the inherent structure, and uses its energy to determine whether to take a step, instead of using the current energy. The algorithm discussed in the previous paragraph would at first lower the energy by using extra dimensions, then it would compress the system back to 3D at constant energy, and only after successful compression would it optimize the energy in 3D.

We conducted several runs with multiple small steps in the walk, varying the length of the component in the fourth dimension. It was set to 0.1, 1 and 2 times the average length of 3D components. The step size was adjusted to 50% acceptance ratio. The results of these calculations are shown in **Figure 18**. The performance is not significantly affected by altering how far the system travels in the 4th dimension. Here is an explanation for this phenomenon. The travel in 4D is done in many small steps. After each step the atomic coordinates are adjusted to keep the system at constant energy. This adjustment must give roughly similar ratios to the 3D and 4D parts of the displacement every time the program is run.

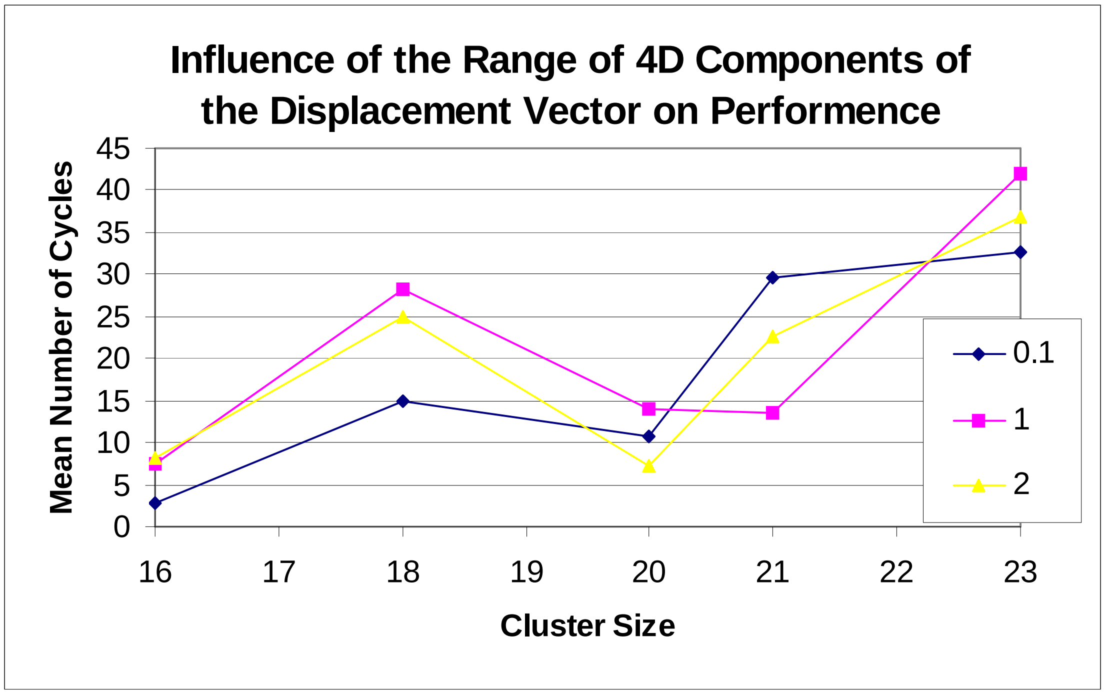


Figure 18 Influence of the range of 4D components of the displacement vector on performance.

Each point in this chart represents the average of nine optimizations of a Lennard-Jones cluster. In all cases the travel was done by taking 100 constant energy steps perpendicular to the gradient of energy. The difference between the runs in the three graphs is the range of the 4th dimensional component of the displacement vector. It was set to 0.1, 1 and 2 times the range of three-dimensional components.

Unfortunately, we did not succeed in significantly improving upon the approach that Faken *et al*. used to move atoms out of a local minimum into 4D space. We found that this step is not very sensitive to details. However, this does not really concern us, because the problem with the compression bottleneck has been solved.

2.2.7. The performance of our algorithm compared with the algorithm of Faken *et al.*

An efficient way to carry out optimization by using constant energy walks in 4D has been developed. This was achieved primarily by significantly accelerating the part of the algorithm that compresses the system from 4D to 3D at constant energy. Attempts were made to improve the step in which the system travels from a 3D local minimum into 4D at constant energy, but this step was not improved significantly.

Figure 19 reports the average numbers of cycles to locate the global minimum of the LJ_{16} through LJ_{39} clusters when using **Equation 38** to accomplish the compression. The figure also indicates the average number of cycles to locate the global minimum for the LJ_{16} through LJ_{30} clusters reported by Faken *et al.* For all clusters for which results are available with both algorithms, the mean number of cycles required to locate the global minimum is smaller with our algorithm. For the $n \leq 26$ clusters, fewer than 20 cycles are required on average to locate the global minimum when compression is done with **Equation 38**. For the larger clusters the number of cycles required to locate the global minimum can be appreciably greater, but even for the $n = 30$ cluster, only 236 cycles (10 minutes on a 1 GHz Pentium III computer) were needed on average to locate the global minimum.

Our implementation of the global optimization algorithm even performed well for the LJ_{38} cluster. Twenty optimizations of LJ_{38} starting from different random arrangements of the LJ atoms were carried out using our implementation of the extended dimensional algorithm. All runs located the global minimum, with the number of cycles required ranging from 374 to 20,657, with the average being 8,367 cycles. In the course of the optimizations, fifteen runs also located the lowest-energy structure in the funnel containing the five-fold minima. Faken *et al.*[47] listed information for only one successful optimization of LJ_{38} which required about 8000 cycles. Thus our implementation of the extended dimensional algorithm performs well even for the difficult case of LJ_{38}. Nonetheless, as is clear from **Figure 19**, there is an overall exponential growth in the number of cycles to locate the global minimum with growing cluster size.

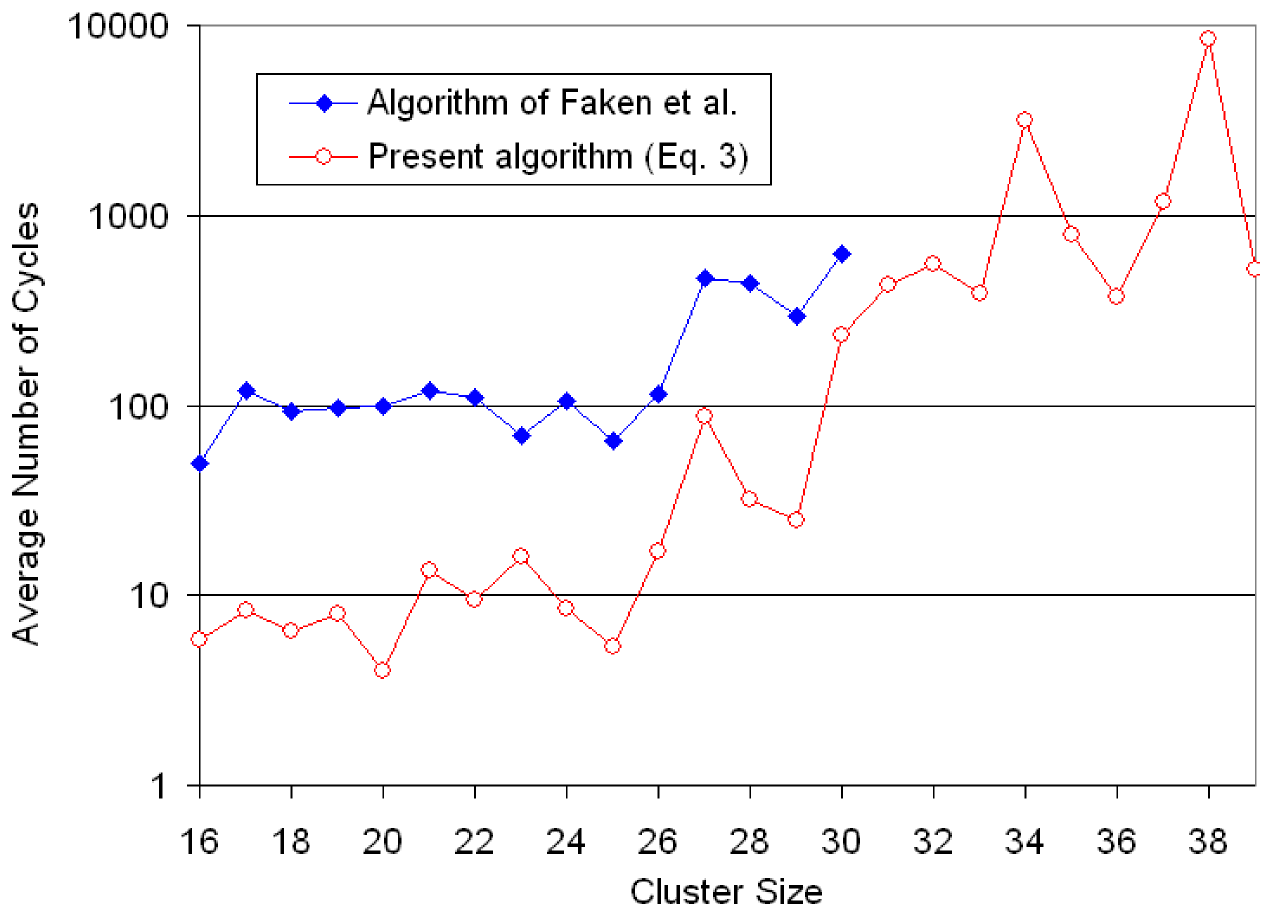


Figure 19 Comparison of performance of our program with program of Faken *et al.*

We optimized each Lennard-Jones cluster nine times, each time starting from a different random configuration. For LJ_{38} we have done twenty optimizations. We report the mean in the number of cycles needed to reach the global minimum.

2.3. CONCLUSIONS

2.3.1. Improvements made to the algorithm of Faken *et al.*

2.3.1.1. Improvements that reduced the number of cycles

We have improved the speed of the algorithm of Faken *et al*. by three orders of magnitude. One order of magnitude improvement was achieved because our implementation of this global optimization algorithm required fewer cycles to find global minima. This is due to our improvements to both the 3D→4D walk at constant energy and to the 4D→3D compression step of the algorithm.

Our implementation of the 3D→4D walk at constant energy differed from that of Faken *et al.*[47] in two respects.

1) In our implementation the same number of steps was used in the 3D→4D walk at constant energy step for all clusters.
2) In cases where the compression step failed to bring the system to 3D, the system was returned to the previous local minimum in 3D. In the algorithm of Faken *et al.*[47] it was brought back to the point in 4D from which the compression was started. Our test revealed that this change generally resulted in a small increase in efficiency in locating the global minimum in 3D. This makes sense, because overall the minima bear some degree of resemblance to each other, and taking the system further and further into 4D makes it look less like a 3D minimum.

2.3.1.2. Improvements to the 4D→3D compression step.

A much more significant acceleration of the algorithm of Faken *et al.*[47] was achieved due to improvements made to the 4D→3D compression step. Faken *et al.* compressed the system from 4D to 3D at constant energy by moving atoms perpendicular to the energy gradient in the direction of decreasing width. (**Figure 6**) We have been able to reduce the CPU time required for the compression step by two orders of magnitude by combining energy and width into a single function (**Equation 37**) and minimizing that function using the algorithm introduced in **Equation 38**. Using this approach also increased the fraction of successful optimizations, because the domain of convergence for this method was greater than that for the approach of Faken *et al.*

2.3.2. Bottlenecks remaining in the algorithm

2.3.2.1. The high number of iterations required for the 4D→3D compression step.

Two bottlenecks still hinder the performance of our algorithm. Although we have improved the speed of the step that compresses the system from 4D to the 3D world by two orders of magnitude, it still takes up more than 99% of the run time.

2.3.2.2. Frequent failures of the 4D→3D compression step.

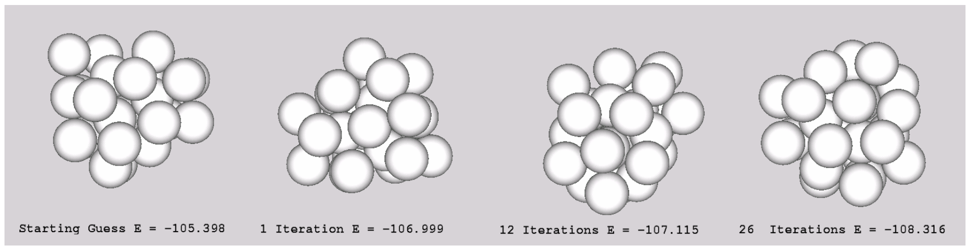

Starting Guess E = -105.398
1 Iteration E = -106.999
12 Iterations E = -107.115
26 Iterations E = -108.316

Figure 20 Progress in the optimization of LJ_{26}.
The longer our algorithm runs the less frequently it finds new minima.

The other problem with the algorithm developed is that the longer it runs, the less frequently it finds new minima that are lower in energy. This is illustrated by **Figure 20** for LJ_{26} and by **Figure 21** and **Figure 22** for LJ_{38}. At first, the algorithm finds minima that are high in energy. It is very easy to find other minima that are lower in energy than those high-energy minima, so after the travel step the compression step frequently succeeds in finding a lower energy minimum. However, as the algorithm gets close to the global minimum, there are very few minima lower in energy than the current minimum. They are hard to find, so after the travel step the compression step usually fails, because when the function containing energy and width is optimized it ends up at a local minimum that is not in the 3D "world". Essentially this algorithm trades the complexity of the 3D surface for the complexity of the constant energy surface in four dimensions.

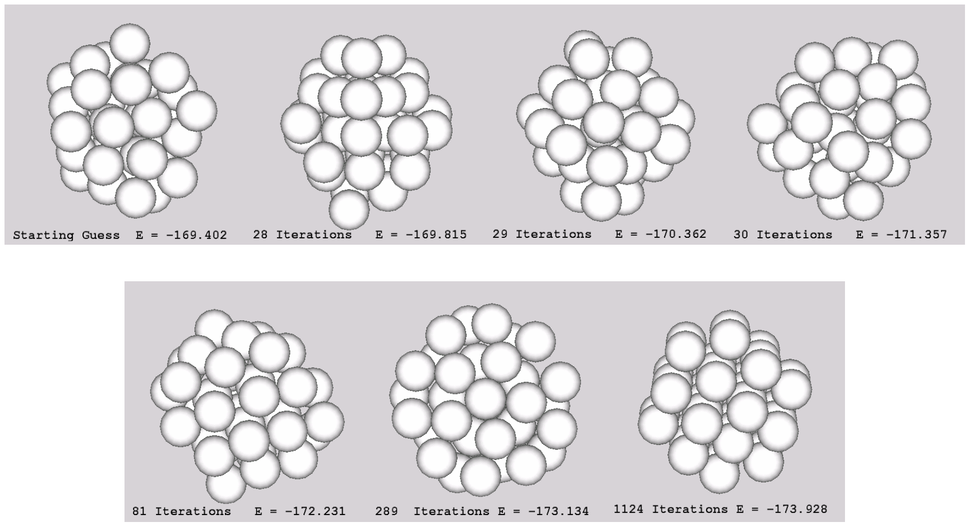

Starting Guess E = -169.402
28 Iterations E = -169.815
29 Iterations E = -170.362
30 Iterations E = -171.357
81 Iterations E = -172.231
289 Iterations E = -173.134
1124 Iterations E = -173.928

Figure 21 Progress in the optimization of LJ_{38}.

The longer our algorithm runs the less frequently it finds new minima.

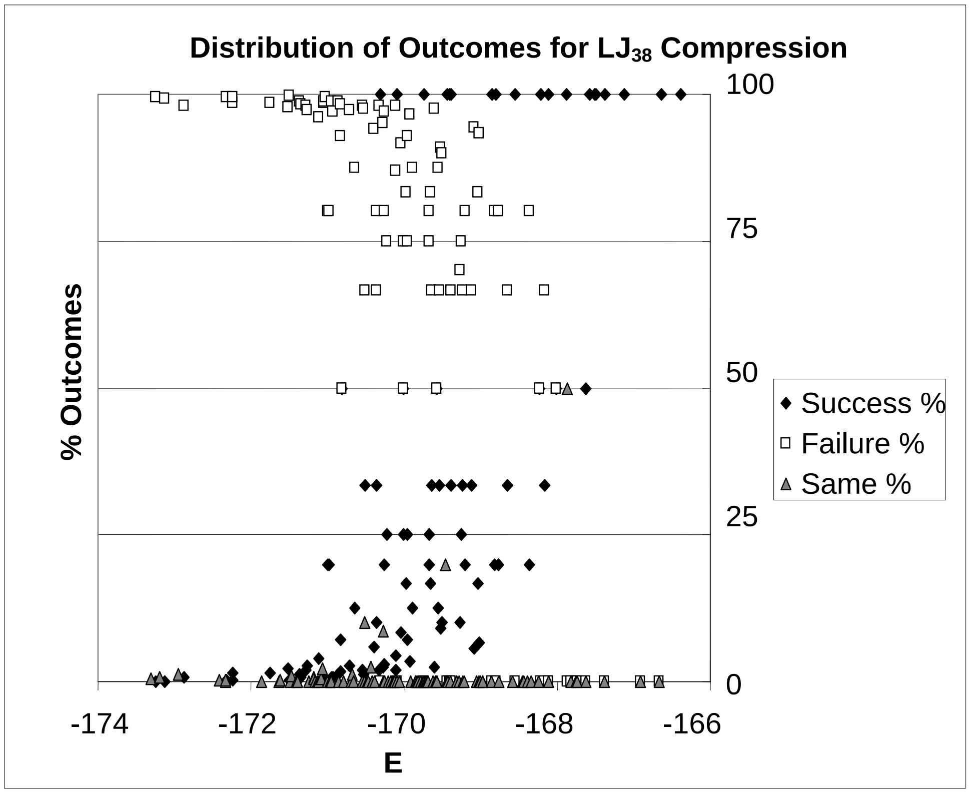


Figure 22 Distribution of outcomes for LJ_{38} compression.

This figure shows how frequently the compression step succeeds when the 4D walk is started from 3D minima of various energies. If the 4D walk at constant energy is started from high-energy minimum, the compression step is likely to succeed. The lower the energy of the minimum from which the walk is started, the more likely is the compression step to fail.

The problem associated with failures of the compression step was not introduced in our implementation. It was already present in the algorithm of Faken *et al.* In fact, our implementation has alleviated this problem to some degree by changing the way we do the travel and the compression step. For every cluster we have tried to optimize, our algorithm required a smaller mean number of steps than the algorithm of Faken *et al.*, meaning that the combination of the travel and compression steps in our implementation of this algorithm succeeds more frequently.

2.3.2.3. Exponential scaling for larger clusters

Faken *et al.* claimed that unlike other global optimization algorithms, their algorithm did not scale exponentially with system size. This is in fact true when the algorithm is used to optimize LJ_{16}-LJ_{25}, however for bigger clusters the mean number of cycles grows exponentially (**Figure 19**).

2.3.3. Possible directions of future research

2.3.3.1. Two approaches to improving the algorithm's efficiency

Although we have improved the performance of the algorithm of Faken *et al.* by three orders of magnitude, that is not enough. To be competitive with modern global optimization algorithms like parallel tempering[36,37,38] and basin hopping,[42] the present algorithm has to be accelerated more. There are two ways to achieve this. One way is to improve individual modules of the present algorithm. Another is to improve the algorithm as a whole.

2.3.3.2. Improving the compression from 4D to 3D at constant energy

Even after being accelerated by two orders of magnitude, the compression from 4D to the 3D "world" at constant energy takes up more than 99% of the run time. This makes it the most logical step to speed up. Although we have tried to improve this step by various means, there might still be better ways of carrying it out. For example, one could experiment with changing the functional form of F(**x**) from one listed in **Equation 37** to

$$F(\mathbf{x}) = (E(\mathbf{x}) - E_0)^4 + W(\mathbf{x}).$$

Equation 39

This should make the canyon-like landscape of F(**x**) flatter on the bottom in the direction of changing E(**x**), and hence make F(**x**) easier to minimize.

2.3.3.3. Improving the algorithm by introducing temperature

Along with improving individual modules one can try to improve the algorithm as a whole. The algorithm of Faken *et al*. has many parallels with Monte-Carlo sampling. The Monte-Carlo algorithm transfers the system between states. The algorithm of Faken *et al*. transfers it between minima. In Monte-Carlo simulations each step can move the system to a new configuration or retain an old one. The same is true for the algorithm of Faken *et al*. If an attempted Monte-Carlo step takes the system to a lower energy state, the step will be taken. If the cycle in the algorithm of Faken *et al*. takes the system to a lower energy minimum, the move to that minimum will be made. The important difference between the two algorithms is that in Monte-Carlo sampling a move that attempts to take the system up in energy is accepted with a Boltzman probability. In the algorithm of Faken *et al*. the system can never go up in energy.

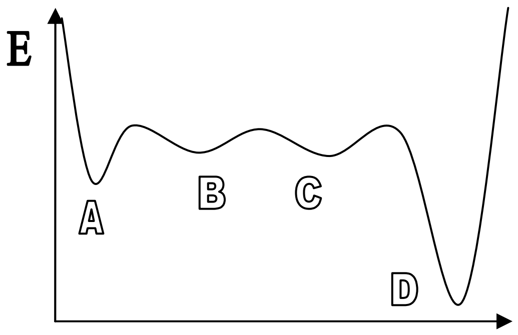


Figure 23 This figure illustrates the advantage of including temperature in extended dimensional optimization.

Suppose one begins the search for global minimum D at a local minimum A. In the current implementation of the algorithm going from A to D can only be accomplished if the 3D→4D travel step takes the system into a region close to D. If this step leaves the system somewhere near B or C, the next step, which compresses the system to 3D at constant energy, will not succeed, because the system will be trapped in a non-zero width local minimum. If temperature is introduced into the algorithm, the compression will be able to proceed to either B or C because they are not much higher in energy than A, and from this new position the next cycle might take the system to its global minimum.

One could introduce temperature into the algorithm of Faken *et al*. Compression to 3D often fails because it runs into local minima of the width function. In the current implementation of the algorithm these configurations are always rejected. One does not have to reject them. One can force them to 3D by setting extra atomic coordinates to 0. Then one could optimize these configurations to 3D local minima, and choose whether to accept these minima by using the Metropolis rules. This opens up the world of Monte-Carlo algorithms, allowing one to create algorithms similar to simulated annealing and parallel tempering[77] that use extended dimensionality.

3. Adsorption of H_2 Molecules on the Si(100) Surface

3.1. Introduction

3.1.1. The Si(100) surface

3.1.1.1. Reconstruction of the Si(100) surface

The Si(100) surface is one of the most important surfaces in the semiconductor industry. Bulk silicon has a diamond-like structure. If a silicon crystal is broken along the (100) Miller pline, each atom of the newly formed surface will have two dandling bonds. The Si(100) surface undergoes reconstruction as shown in **Figure 24**. The neighboring surface atoms pair up to form dimers.[57] Although the Si-Si bond in these dimers has some π bond character, its comparison with the double bond in Si_2H_4 indicates that the bond order in the dimer bonds is closer to one than two.[58]

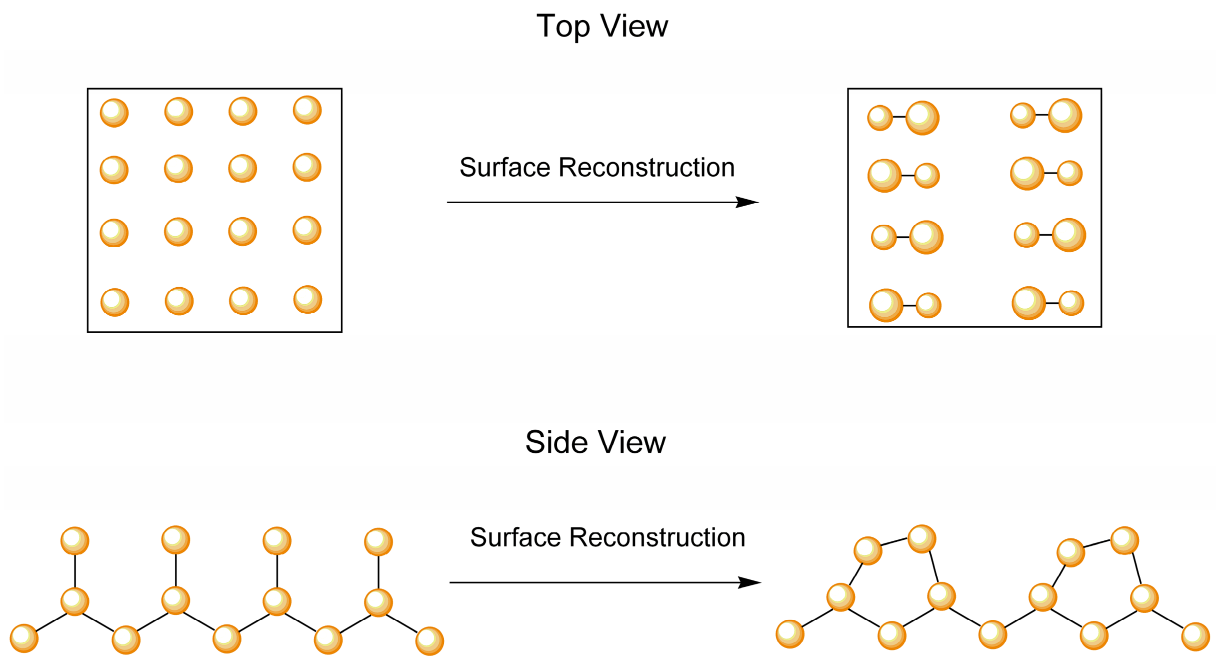


Figure 24 Reconstruction of the Si(100) surface.

Images of the Si(100) surface taken using Scanning Tunneling Microscopy (STM) and Low Energy Electron Diffraction (LEED) show the dimers to be arranged in long, parallel rows on the surface.[59,60,61,62]

3.1.1.2. Buckling of dimers on the Si(100) surface—a controversial issue

Although the Si(100) surface has been a subject of numerous experimental and theoretical studies, it is still not clear if the Si-Si dimers that form on that surface are flat as shown in **Figure 25** or buckled as shown in **Figure 26**. Part of the reason for this is the very flat nature of the potential energy surface along the buckling coordinates. Small variations in experimental conditions[63,64,65,66,67,68] and the use of different theoretical methods lead to contradictory results on the structure of these dimers.

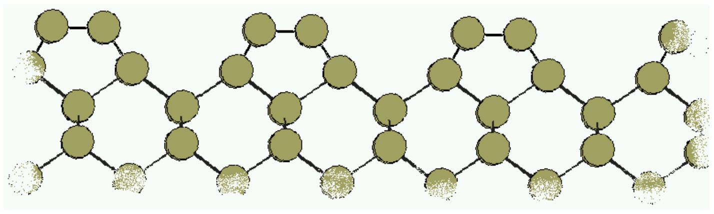


Figure 25 Flat dimers on the Si(100) surface.

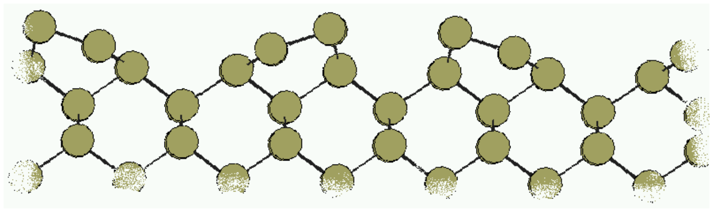


Figure 26 Buckled dimers on the Si(100) surface.

High-level multireference wave function methods predict that the dimers on the Si(100) surface are flat,[69,70,71,72,73] while DFT and quantum Monte Carlo (QMC) predict that they are buckled.[74,75,76] Jung *et al.* believe that the dimers on the Si(100) surface are flat at some temperatures and buckled at others.[77]

3.1.2. An STM and STS study of H_2 adsorption by the Si(100) surface

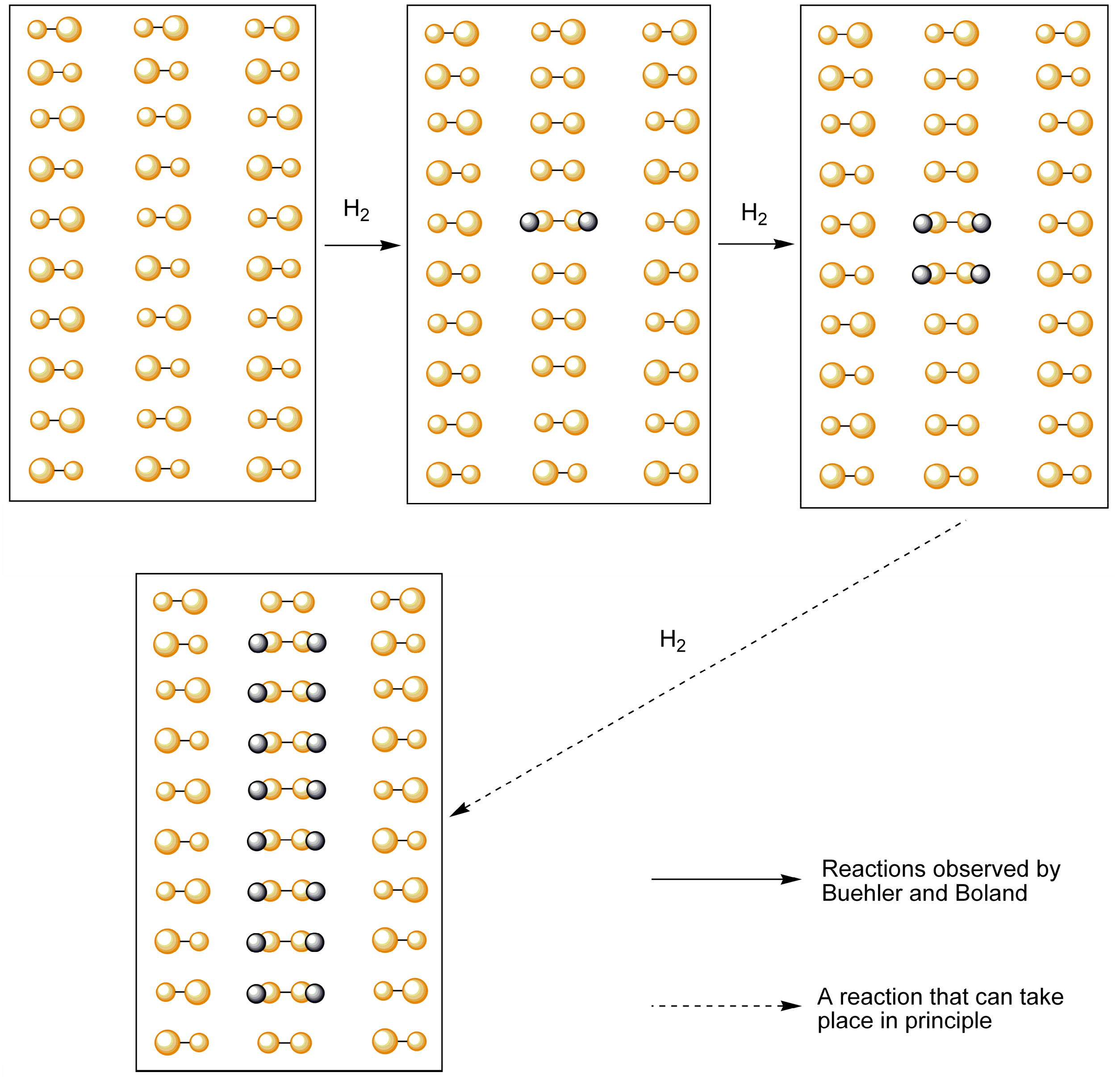

H2
H2
H2
Reactions observed by Buehler and Boland
A reaction that can take place in principle

Figure 27 H_2 adsorption in the Si(100) surface as observed by Buehler and Boland and a possible way to build linear nanostructures.

Using STM and scanning tunneling spectroscopy (STS) Buehler and Boland explored how adsorption of H_2 molecules influences the topology and reactivity of the Si(100) surface.[78] These experiments suggest that adsorption of a hydrogen molecule onto a Si-Si dimer induces unbuckling of nearby dimers in the same dimer row. This dramatically changes the reactivity of the dimer adjacent to the site where H_2 is adsorbed, increasing the rate of H_2 adsorption by a factor of a billion. This means that once one H_2 molecule is adsorbed onto the Si(100) surface, other H_2 molecules will adsorb in the same row, which could provide a way to build linear nanostructures on the Si(100) surface.

3.2. Experimental

In order to address the issue of the influence of adsorption of H_2 molecules on some of the Si-Si dimer sites on the buckling of nearby bare SiSi dimer sites, a series of DFT calculations using different size supercells and different hydrogen coverages were carried out. The calculations were performed with the Vienna Ab Initio Simulations Package (VASP)[79,80,81,82] and made use of slab models with periodic boundary conditions, the gradient-corrected PW91 functional,[19] ultrasoft (Vanderbilt-type) pseudopotentials,[83,84] a plane-wave basis set with a 200 eV cutoff, and three Monkhorst k points.[85] The supercells contained two, three, or four surface dimers (in a dimer row) and five layers of silicon atoms. The Si atoms in the bottommost layer were kept frozen in their bulk positions and terminated with H atoms to eliminate the dangling bonds. The positions of the Si atoms in the top four layers and of any adsorbed H atoms were fully optimized. A 9 Å vacuum layer was introduced between adjacent slabs.

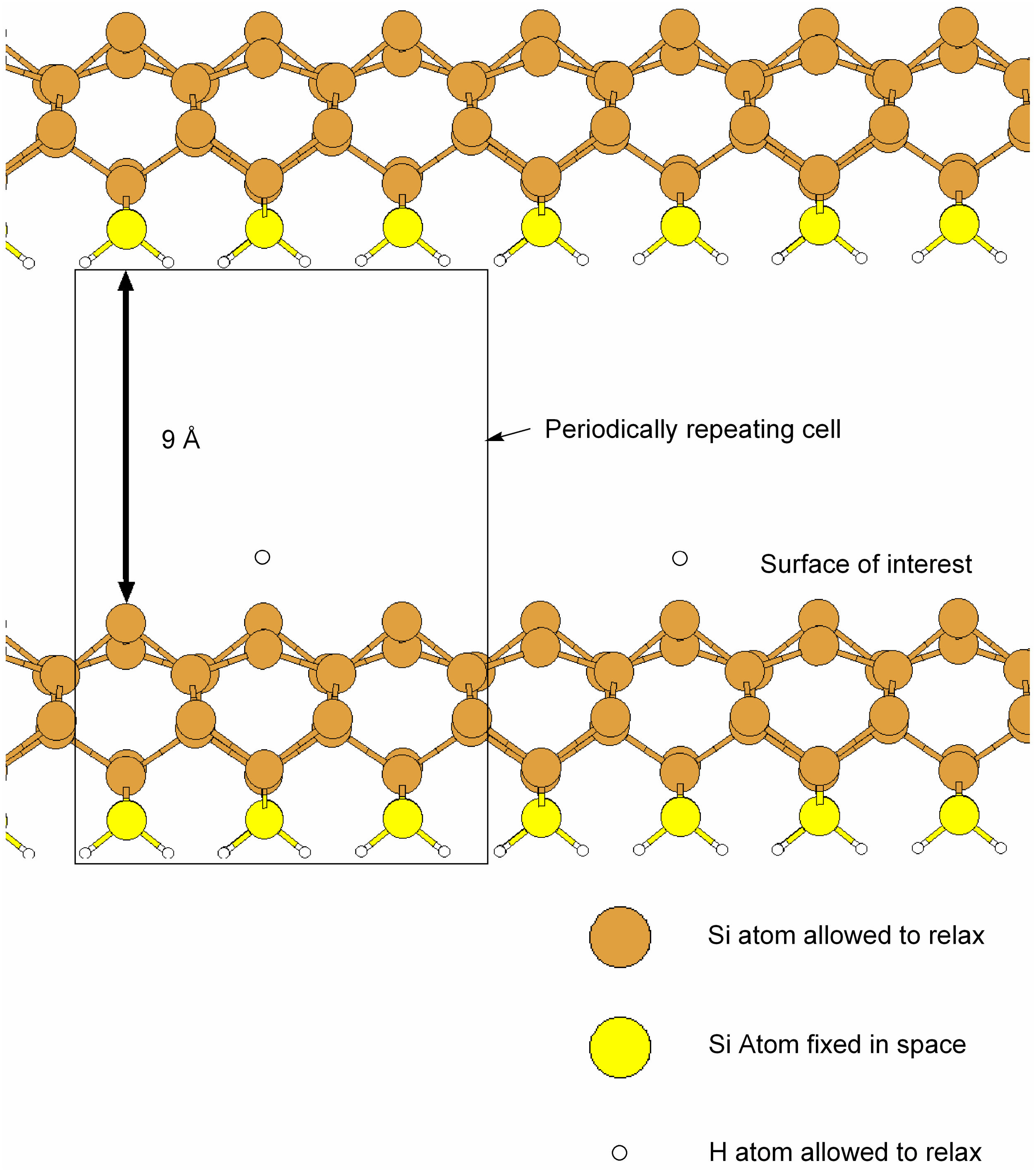


Figure 28 The supercell used in calculations.

3.3. Results

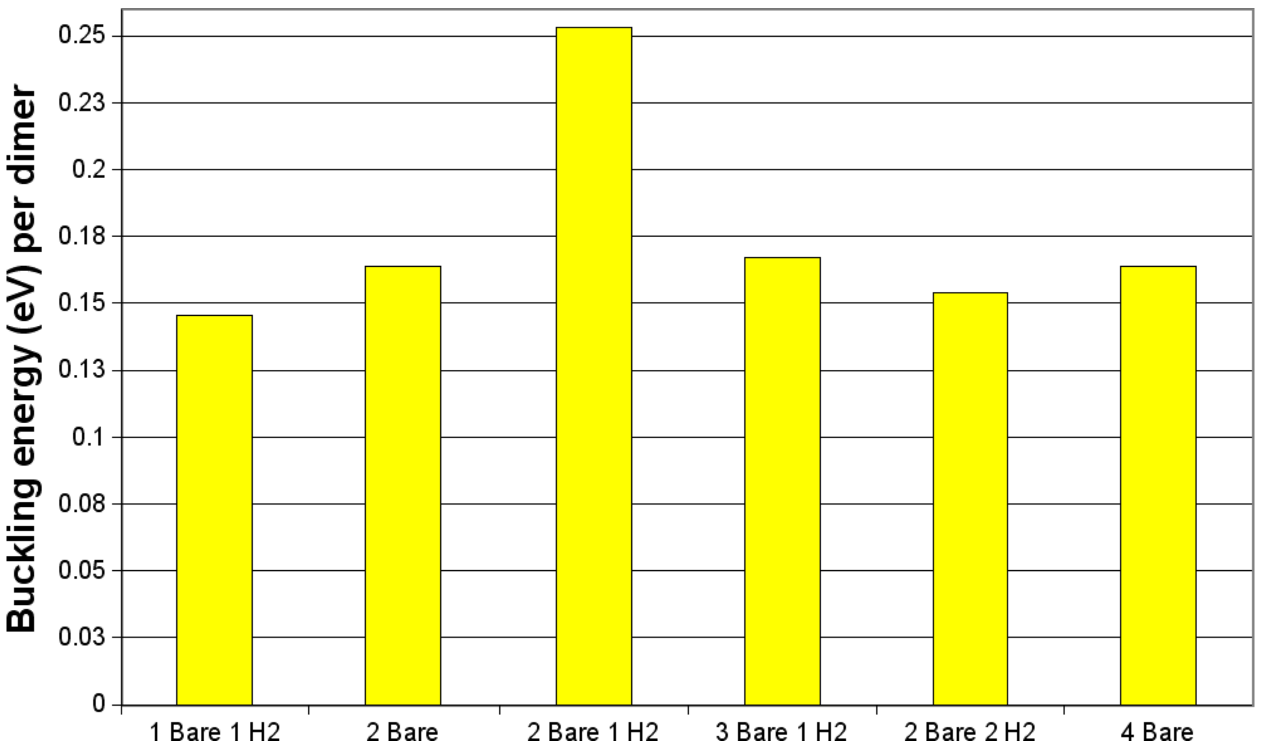


Figure 29 Influence of the adsorption of H_2 on the energetics of buckling of bare SiSi dimers on the Si[100] surface as predicted by DFT calculations with PW91 functional.[19]

The results of the calculations are summarized in **Figure 29** and **Figure 30**. From the former figure it is seen that the buckling energy per surface dimer is calculated to be the same (-0.16 eV) in the two- and four-dimer models. (Results are not reported for the three-dimer model since a *trans* pattern of buckling of adjacent dimers cannot be attained in that case.) For the two- and four-dimer models occupation of a subset of the available dimer sites by hydrogen has little impact at the tendency of the remaining sites to buckling, the buckling energies ranging from -0.15 to -0.17 eV per dimer. Interestingly, in the three-dimer model, with one dimer site occupied by hydrogen, the buckling energy per dimer is -0.25 eV. The origin of the extra stability associated with buckling is not clear in this case.

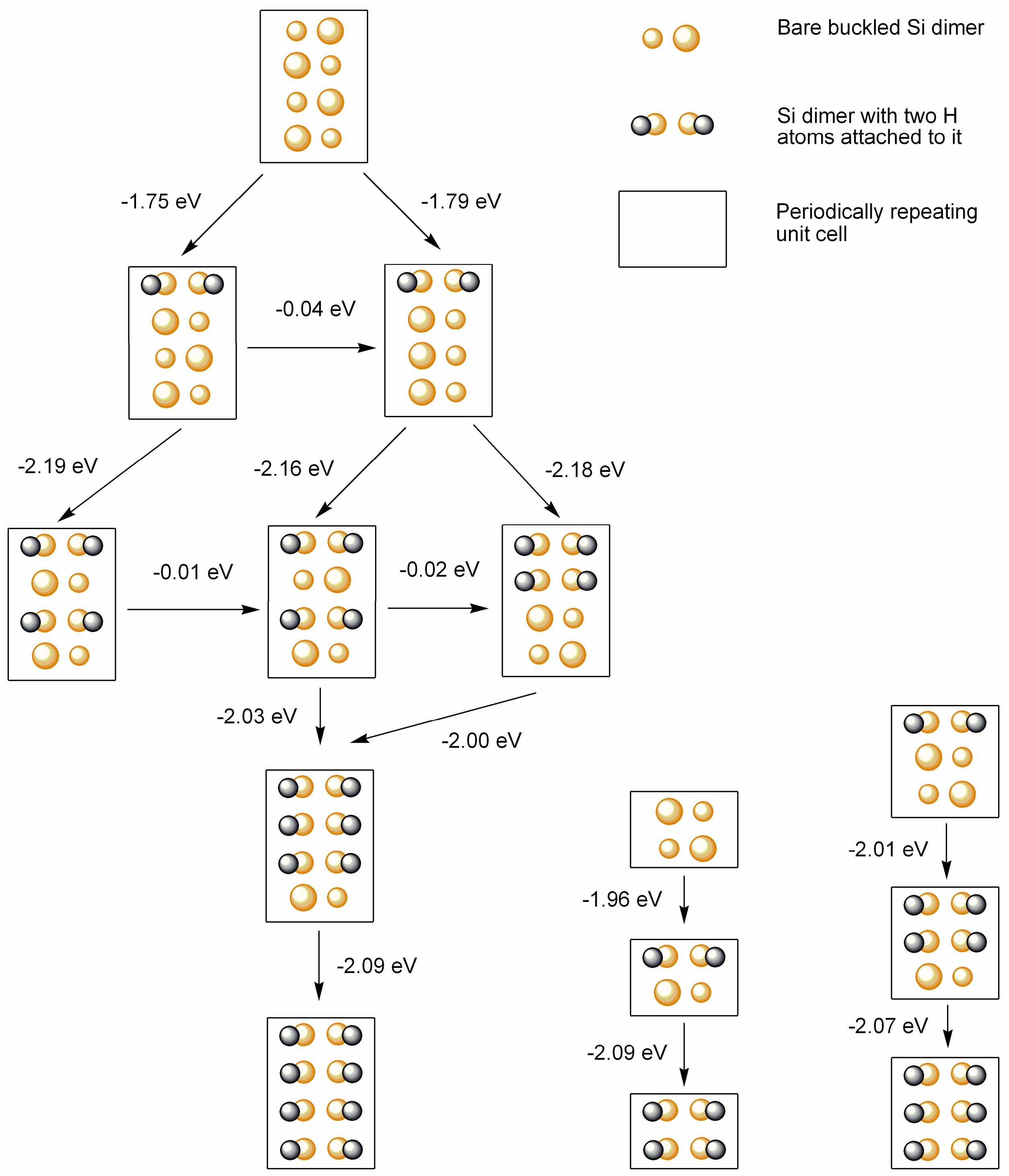


Figure 30 Dependence of the energetics of H_2 adsorption on the Si[100] surface on the coverage as determined from DFT calculations with the PW91 functional.[19]

Figure 30 examines the coverage dependence of the H_2 adsorption energies. The energy change associated with the adsorption of an H_2 molecule onto a dimer is calculated to be -1.93 and -1.97 eV in the two-dimer and four-dimer models, respectively. The adsorption of additional H_2 molecules is calculated to be 0.07 - 0.15 eV more favorable than the adsorption on a bare surface, with the precise enhancement depending on the distance from already occupied sites and the extent of hydrogen coverage.

3.4. Conclusions

Some of our results are consistent with the experimental data of Buehler and Boland, but others are not. Consistent with the experiments, the calculations predict an energetic preference for clustering of occupied sites in a dimer row. However, our calculations did not verify the unbuckling induced by H_2 adsorption reported by Buehler and Boland. In fact, in some cases our calculations predicted that the presence of adsorbed hydrogen makes unbuckling less energetically favorable. (See **Figure 29**.) It is possible that the unbuckling observed by Buehler and Boland was caused by the presence of the STM tip.

Our calculations have confirmed that building linear nanodevices could be possible by H_2 adsorption on the Si(100) surface.

4. Computational Insight Into the Regiocontrol of the Molybdenum and the Rhodium Catalyzed [2 + 2 + 1] Cyclocarbonylation Reaction

4.1. Introduction

4.1.1. The antibiotic resistance problem

One of the major problems in modern medicine is the emergence of antibiotic-resistant bacteria. Normally if regular antibiotics do not work, one uses vancomycin. Vancomycin is not a perfect antibiotic, though, as enterococcus bacteria are quickly developing resistance to it. According to the Centers for Disease Control, from 1989 to 1993 the percentage of nosocomial infections caused by VRE (vancomycin-resistant enterococcus) increased from 0.3% all the way to 7.9%, a 34-fold increase in only four years.[86] That is why a lot of effort is going into development of antibiotics that will work on VRE.

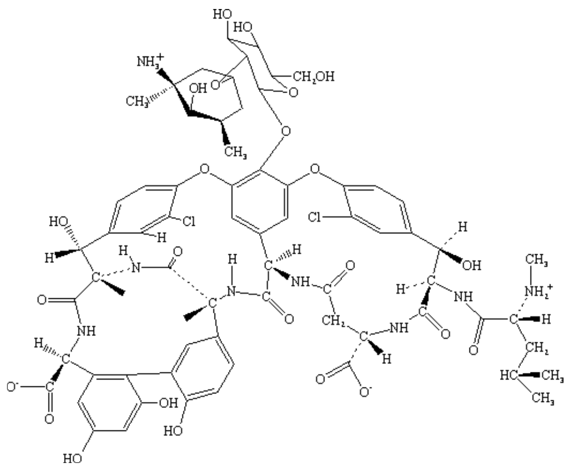


Figure 31 Vancomycin

Vancomycin is an antibiotic used in the prophylaxis and treatment of infections caused by Gram-positive bacteria. It is a branched tricyclic glycosylated nonribosomal peptide produced by bacterial fermentation.

All experiments described in this chapter were carried out in Prof. Brummond's group unless noted otherwise.

4.1.2. Guanacastepene A

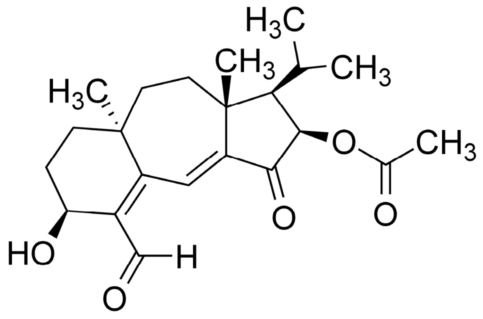


Figure 32 Guanacastepene A, a natural product with a promising antibiotic activity.

Guanacastepene A is a compound showing promising antibiotic activity against vancomycin-resistant bacteria.[87] This natural product, shown in **Figure 32**, was isolated by John Clardy *et al.*[88] It was extracted with hexane from a Costa Rican fungus (CR115) found on the branches of the *Daphnopsis Americana* tree (**Figure 33**) and purified by chromatography.

Figure 33 *Daphnopsis Americana*[89] tree commonly known as "burn nose".
Guanacastepene A was isolated from a fungus found on the branches of a tree of this species.

Professor Brummond's group at the University of Pittsburgh is trying to find a synthetic route to guanacastepene A and its analogs.[90] The key transformation in this synthesis is shown in **Figure 34**.

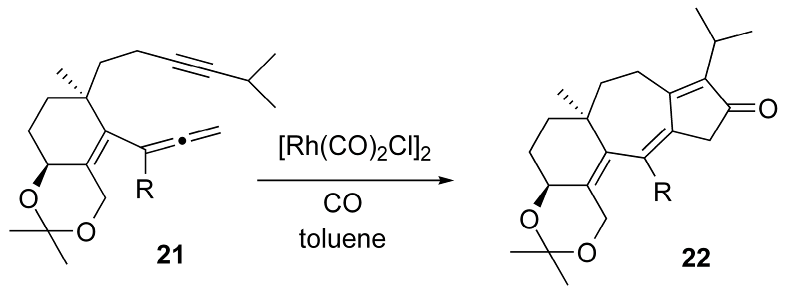


Figure 34 An important step in the synthesis of guanacastepene A.

The unique feature of this step is the simultaneous formation of the 7-5 ring system by the selective reaction of the distal double bond of the allene in a [2 + 2 + 1] cycloaddition reaction. Experiments carried out on a similar system[91] indicate that other transition-metal complexes, such as $Mo(CO)_6$, produce a 6-5 ring system resulting from a selective reaction with the proximal double bond of the allene.

The transformation of **21** to **22** has been made possible by using rhodium biscarbonyl chloride dimer. The goal of this project is to understand the unusual transition-metal directed selectivity.

4.2. Discussion and Results

4.2.1. Experimental

4.2.1.1. Basis set and level of theory

All calculations were performed with the GAUSSIAN03 package[92] using density functional theory with Becke's three-parameter hybrid exchange functional and the Lee-Yang-Parr correlation functional (B3LYP).[20] The 6-31G(d) basis set[93] was used for H, C and O atoms and the double-ζ quality, Hay and Wadt LANL2DZ basis set was used in conjunction with the LANL2 effective core potential[94] for Rh. All calculations were done in a vacuum.

4.2.1.2. Transition state searching

Transition states were found using QST3,[95] QST2,[96] and coordinate-driven methods. Some transition states seem to connect one minimum with two alternative minima. In such cases there should be additional stationary points on the pathway lower in energy than the identified transition state. These "missing" stationary points are not relevant for determining the overall activation energies.

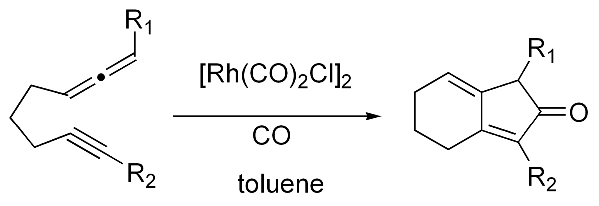
4.2.1.3. A comment about solvents

Experiments were carried out in 1,2-dichloroethane and toluene. Neither solvent is expected to significantly impact the reaction pathway. Therefore solvent was not included in calculations.

4.2.2. Modeling Rh-catalyzed intermolecular [2 + 2 + 1] cyclocarbonylation reaction

We needed some information about the barriers involved in various reaction steps in order to know which steps we should model most extensively, so we performed a series of calculations on an intermolecular cyclocarbonylation reaction. These were not intended to give us detailed and accurate information, but rather to aid us in planning further more accurate calculations. Unlike the calculations in Chapter 3, the calculations on Rh-catalyzed intermolecular [2 + 2 + 1] cyclocarbonylation reaction included both the electronic energies of the minima and of the transition states, but were less rigorous than our subsequent calculations. They mapped only one reaction pathway – the one that was the most intuitive to us. These calculations were of the electronic energies only, while the subsequent calculations were of free energies.

4.2.2.1. The choice of a model system



R_1=H, R_2=TMS
R_1=Ph, R_2=TMS
R_1=*t*-Bu, R_2=H

Figure 35 Rh-catalyzed reactions leading to the formation of the 6-5 ring system.

Because rhodium-based catalysis can form both the 7-5 ring system[90,97] (**Figure 34**) and the 6-5 ring system[97] (**Figure 35**), we concluded that the size of the larger ring system was not important to the reaction mechanism and we decided to model the reaction without the larger ring system, the intermolecular condensation reaction between 2,3-pentadiene, 2-butyne and carbon monoxide shown in **Figure 36**. An added bonus was the fact that intermediate **45**, shown in **Figure 38**, did not have a conformational space as large as analogous intermediates, where the allene and the alkyne are connected by a tether.

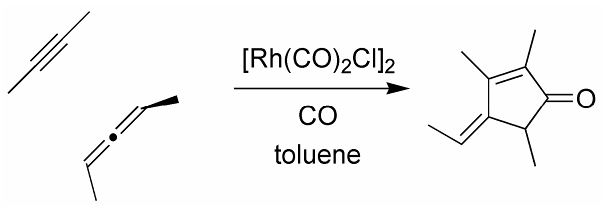
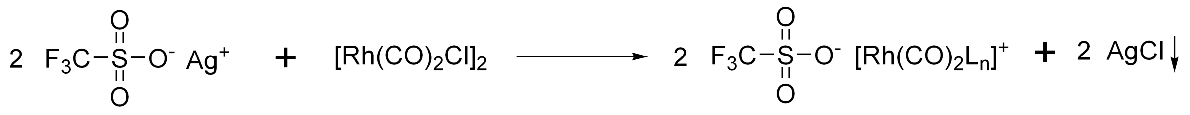


Figure 36 The intermolecular condensation reaction between 2,3-pentadiene, 2-butyne and carbon monoxide. We used this as a model reaction for rhodium-catalyzed carbocyclization.

4.2.2.2. Assumptions about the intermolecular cyclocarbonylation reaction

When modeling of the intermolecular cyclocarbonylation reaction was started, three important assumptions were made. First was that chloride ligands were not involved in the reaction. This assumption was dictated by experimental results for analogous reactions. The addition of silver cations to the reaction mixture did not hinder the reaction. In fact, the addition of silver triflate increased the reaction rate. Silver cations have a very strong affinity for chloride. The addition of silver triflate to the solution removed all the chloride ligands from rhodium centers via precipitation of silver chloride (**Figure 37**).



2 $F_3C-S(=O)_2-O^-$ Ag^+
+
$[Rh(CO)_2Cl]_2$
2 $F_3C-S(=O)_2-O^-$ $[Rh(CO)_2L_n]^+$
+
2 AgCl

Figure 37 Reaction of silver triflate with $[Rh(CO)_2Cl]_2$.

It was also assumed that a monomeric rhodium species catalyzes the reaction. This is a safe assumption. If there are no chloride ligands on rhodium in this reaction, this metal center has an overall charge of +1. Proximity of two positively charged centers in a non-polar solvent is highly unfavorable energetically because of electrostatic repulsion.

By analogy with known mechanisms for other organorhodium reactions[98] it was assumed that there were two carbonyl ligands bonded to rhodium. Another reason we assumed that no more than two carbonyl ligands were involved in the carbocyclization step was that in analogous reactions[99] when CO was not bubbled through the solution, the carbocyclic ring system still formed and there were only two carbonyl ligands per rhodium in the catalyst used.

4.2.2.3. Modeling individual steps of the intermolecular condensation reaction

The first few steps of the reaction were not modeled. It was assumed that rhodium(I) has a high affinity for double and triple C-C bonds, so the attachment of rhodium biscarbonyl to 2,3-pentadiene and 2-butyne did not require overcoming significant barriers.

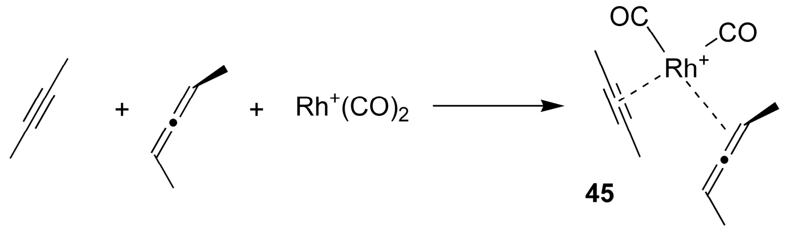


Figure 38 Attachment of rhodium biscarbonyl to 2,3-pentadiene and 2-butyne.

Modeling of the reaction pathway was started with intermediate **45**. This intermediate had several isomers. We started with the isomer **45c1** and converted it to **50b3** by moving C3 and C7 together. Overcoming the barrier for this reaction step required 18.9 kcal/mol of electronic energy. The transition state was clearly an early transition state. Structurally it resembled the reactant very closely. The product of this reaction step, intermediate **50b3**, had a geometry very different from **45c1** and the transition state. In it rhodium no longer had a square planar geometry. This reaction step was highly exothermic. The product was 19.8 kcal/mol below the reactant. Because the electronic energy barrier for the reverse reaction was 38.8 kcal/mol, we concluded that this step was irreversible.

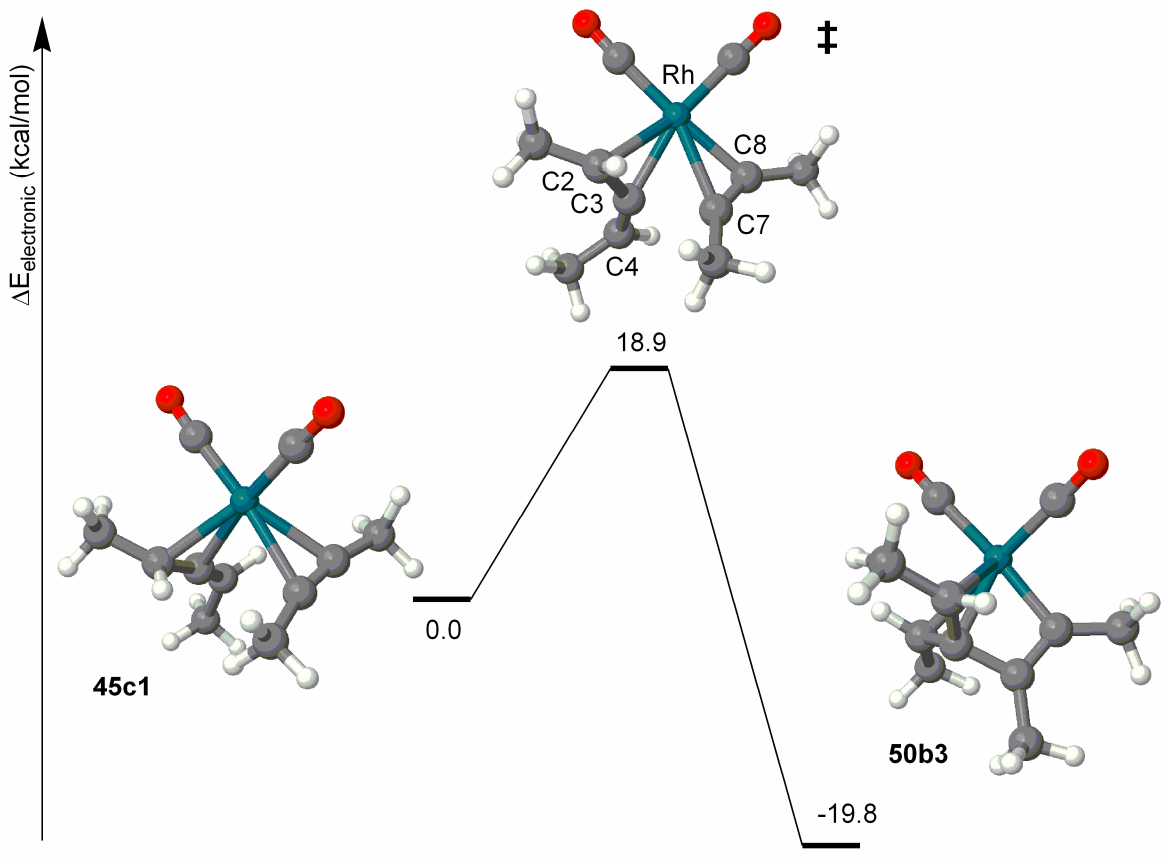


Figure 39 The product-determining step of the Rh-catalyzed intermolecular condensation reaction. The transition state is an early transition state. It resembles the reactant both structurally and energetically.

In the next step of the reaction a carbon monoxide molecule from the media attaches to intermediate **50** (**Figure 40**). It was assumed that this process does not require overcoming a significant barrier. (This assumption was later confirmed by calculations on carbonyl insertion into **127**.)

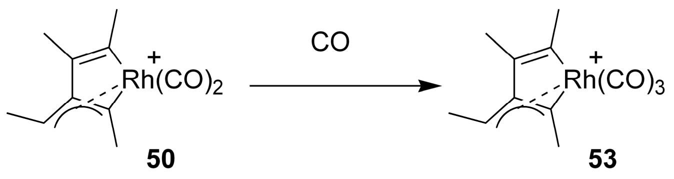


Figure 40 CO adsorption by intermediate 50.

The geometry of intermediate **53** was obtained by adding a carbonyl group to the vacant position on the rhodium atom of intermediate **50b3** and optimizing the resulting structure. The geometry of **53** was very similar to that of **50b3**. CO adsorption by intermediate **50** was found to be an exothermic process with the decrease in the electronic energy being 22.5 kcal/mol.

The next step in the catalytic pathway was carbonyl insertion. There were two possible positions where a carbonyl group from rhodium could insert into the ring. It could insert between Rh and C8 forming metallocyclic intermediate **54**, or between Rh and C4 forming either **55** or **56**, which differ energetically. Metallocycle **56** had an η^2 bond between C2, C3 and Rh. This bond was not present in **55**.

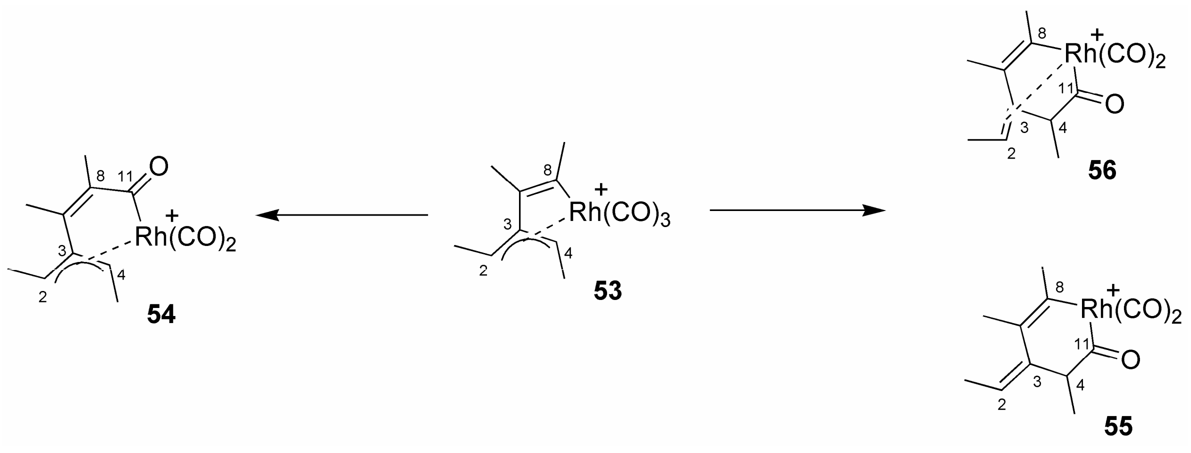


Figure 41 Three possibilities for CO insertion into the ring system.

At first the formation of **54** was considered. It was modeled by starting with intermediate **53** and bringing C11 and C8 together. This process required 19.6 kcal/mol to overcome the barrier. The transition state for this reaction step is shown in **Figure 42**. This process was highly exothermic. The electronic energy dropped by 21 kcal/mol relative to **53**. The step was also irreversible. It would take 40.6 kcal/mol to overcome the barrier for the reverse reaction.

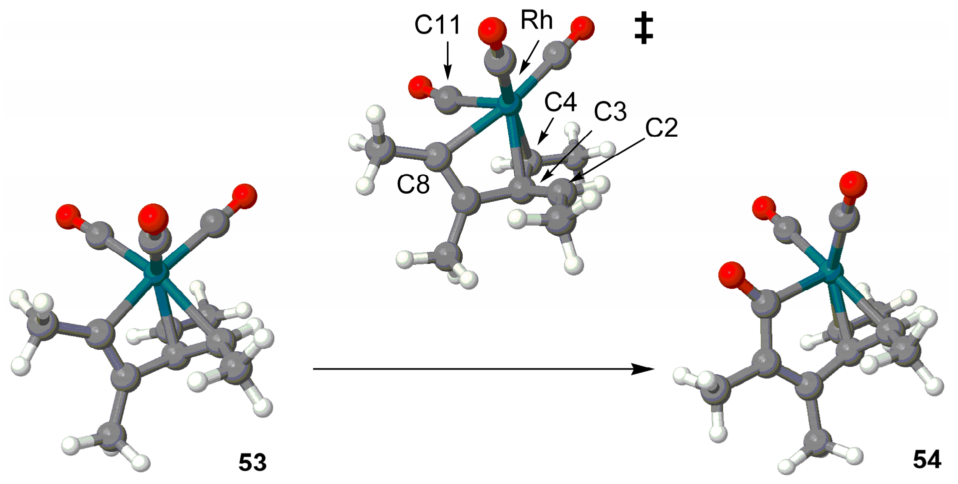


Figure 42 Intermediate 53 converting to 54 by transferring a CO from rhodium into the ring system.

The possibility of CO insertion on the other side of the ring was also considered. Modeling of this process was started with **53**. Decreasing the distance between C4 and C11 led to the formation of intermediate **56** (**Figure 41**). The height of the barrier for this process was 25.1 kcal/mol. This reaction step was not exothermic as with formation of intermediate **54**. Intermediate **56** was above **53** in electronic energy by 6.9 kcal/mol.

The formation of **54** is preferred over the formation of **56** for several reasons. In **54** the double bond between C7 and C8 is conjugated with the carbonyl group; in **56** it is not. In **54** Rh, C3, C7, C8 and C11 form a 5-membered ring. In **56** the η^2 bond between C2, C3 and Rh results in the presence of two 4-membered ring systems, Rh-C3-C4-C11 and Rh-C3-C7-C8. Unlike 5-membered ring systems, 4-membered rings are high-energy structures. The η^2 bond in **56** also turns the C2-C3 double bond at an 88.4° angle to the C7-C8 double bond. This prevents conjugation between these two adjacent double bonds.

4.2.2.4. The mechanism of Rh-catalyzed intermolecular cyclocarbonylation reaction

The proposed mechanism of the [2 + 2 + 1] intermolecular cyclocarbonylation reaction is shown in **Figure 43**. In this mechanism rhodium has two CO ligands attached to it most of the time. This came from analogy with other known rhodium catalysis pathways and experiments in which CO was not bubbled through a solution. Our current understanding of the energetics of most of this pathway is shown in **Figure 44**. The rate determining step is the oxidative addition that leads to the formation of metallocycle **50**.

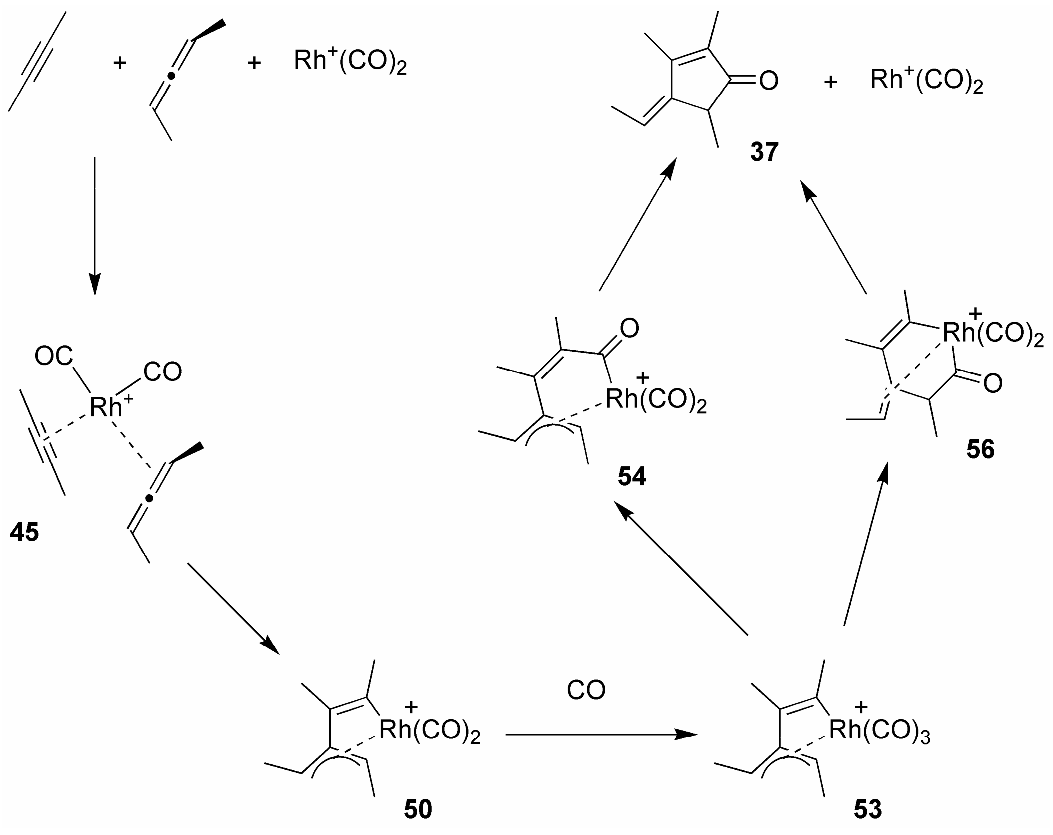


Figure 43 Proposed mechanism of an intermolecular cyclocarbonylation reaction.

At first 2,3-pentadiene and 2-butyne bind to a rhodium complex, then a rhodium metallocycle forms, followed by a CO transfer from rhodium into the ring. This is followed by reductive elimination of rhodium from the ring.

The existence of low-energy metallocycles **50** and **53** helps to determine the product of the reaction. In these intermediates 2-butyne binds to the third carbon of 2,3-pentadiene. We also studied metallocycle **40**, where 2-butyne binds to the second carbon of 2,3-pentadiene (**Figure 45**). This metallocycle was 11.8 kcal/mol higher in energy than **50**. The energy difference is caused by two factors. In **40** the organic ligand binds to the rhodium complex with two η^1 bonds, and in **50** it binds to rhodium with one η^1 and one η^3 bond (**Figure 45**). The other factor in the stability of **50** is that the organic ligand is conjugated. In fact, in intermediate **41** (**Figure 45**), which can also in principle lead to the correct products, the organic ligand binds to rhodium with two η^1 bonds just as in intermediate **40**. It is only 6.3 kcal/mol lower in energy than **40**. This shows that the η^3 bond is not the only important factor in determining the reaction outcome.

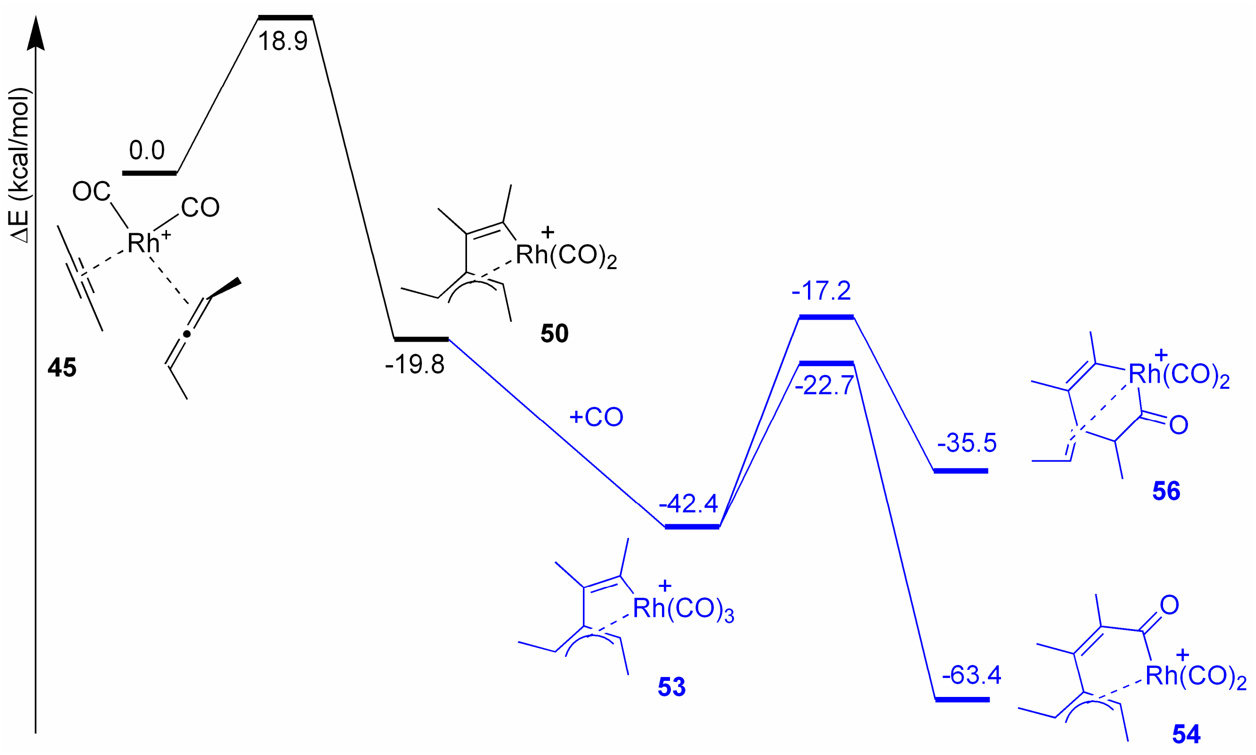


Figure 44 Energetics of intermolecular condensation reaction between 2,3-pentadiene, 2-butyne and carbon monoxide.

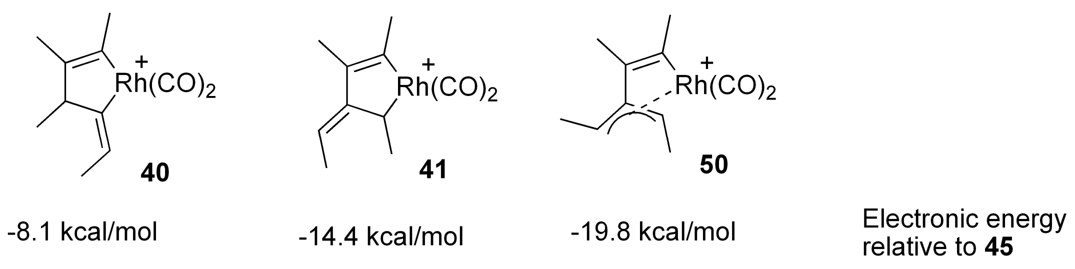


Figure 45 Three intermediates important to our reasoning: 40, 41 and 50.

Intermediate 40 does not lead to analogs of products observed in experiments. Intermediate 41 is more stable because it has conjugation. At first we thought that the reaction proceeded through this intermediate, but now we believe that it proceeds through intermediate 50, which leads to the correct product analog just as 41, but in addition to being stabilized by conjugation it is also stabilized by an η^3 bond.

4.2.3. Modeling [2+2+1] intramolecular cyclocarbonylation reactions

From calculations on the intermolecular cyclocarbonylation reaction we obtained an overall picture of the energetics of the rhodium-catalyzed [2 + 2 + 1] cyclocarbonylation reaction of allenes. With this information we proceeded to study systems that closely resembled the crucial step in synthesis of guanacastepene A (**Figure 34**).[90]

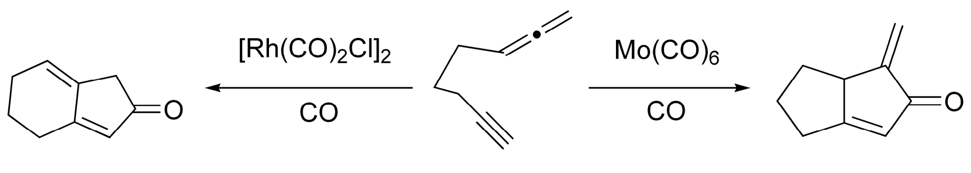


Figure 46 The model system for intramolecular [2 + 2 + 1] cyclocarbonylation.

Our model system is shown in **Figure 46**. If the [2 + 2 + 1] cyclocarbonylation reaction of octa-1,2-dien-7-yne in the presence of carbon monoxide is catalyzed by a Rh(I)-based catalyst, a [4.3.0] bicyclic ring system is formed, but if this reaction is promoted by a molybdenum-based catalyst such as $Mo(CO)_6$, a [3.3.0] bicyclic ring system is formed instead.[91]

Starting with our data pertaining to the rhodium-catalyzed intermolecular [2 + 2 + 1] cyclocarbonylation reaction, we have modeled both the rhodium-catalyzed and the molybdenum-promoted cyclocarbonylation reactions.

The modeling was done by analogy with the intermolecular cyclocarbonylation reaction. It was assumed that the step associated with the highest-energy transition state is the oxidative addition step to form the metallocycle. This step probably determines which ring system is formed, and thus received most of our attention. Further steps such as the addition of an extra CO to the rhodium center, CO insertion and reductive elimination were modeled less rigorously, because they were assumed to require smaller energy barriers or follow irreversible steps. This assumption turned out to be true for the rhodium-catalyzed but not for the molybdenum-promoted cyclocarbonylation reaction. After this, various alternative pathways were modeled.

By analogy with known reaction mechanisms it was assumed that through most of the pathway the molybdenum atom has three carbonyl groups on it and the rhodium atom has two.

4.2.4. Modeling molybdenum-promoted intramolecular [2 + 2 + 1] cyclocarbonylation

4.2.4.1. The oxidative addition step

It was assumed at the outset that the oxidative addition was the product-determining step for the molybdenum-promoted cyclocarbonylation reaction, so it was modeled first. Information on subsequent steps was obtained using results of the oxidative addition calculations.

Starting with intermediates **131c1**, **131c2**, **131c3** (**Figure 47**) and **126c1** (**Figure 48**) and bringing the appropriate carbon atoms together resulted in the formation of compounds **129c1** and **132c3** (**Figure 48**). In addition, attempts were made to identify more transition states by forcing appropriate carbon atoms apart leading back to the reactants; however, that did not yield any new transition states.

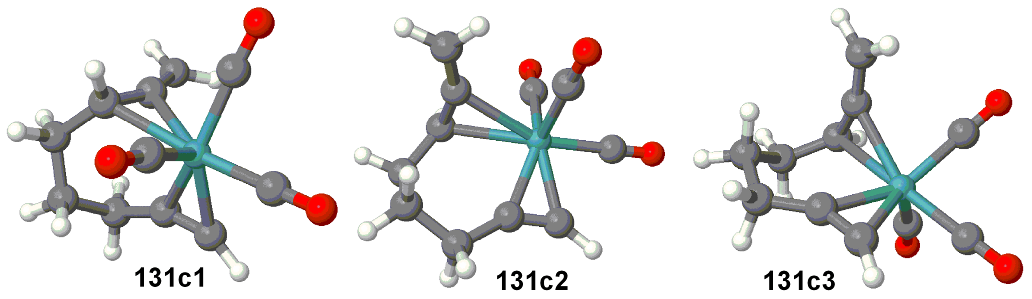


Figure 47 Isomers of Mo-containing intermediate 131.

All low-energy complexation systems involved Mo coordinated to the proximal double bond of the allene; compare **131c3** and **126c1** in **Figure 48**. Similarly, the transition states involving the proximal double bond are all lower in free energy than those involving the distal double bond (compare the transition state between **131c3** and **129c1** to the one between **131c3** and **132c3** in **Figure 48**).

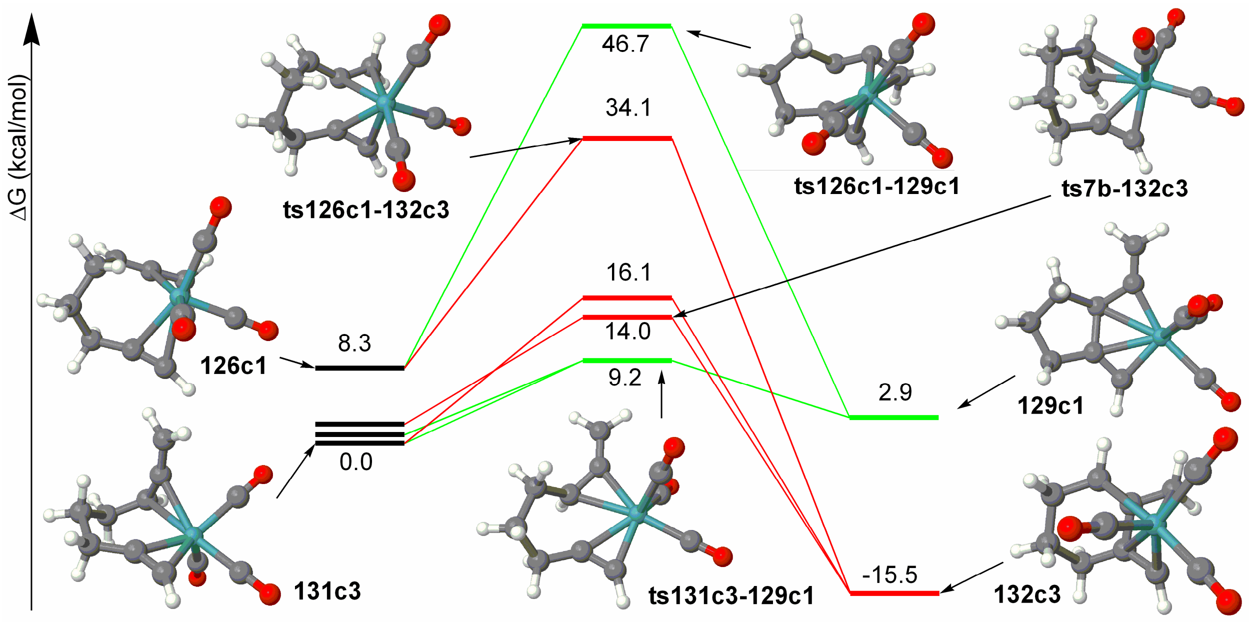


Figure 48 Energetics of the Mo-promoted oxidative addition.

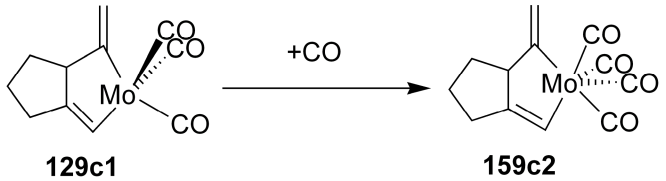
Pathways that lead to the experimentally observed product are shown in green and those pathways that lead to the product that is not formed are shown in red.

The free energy barrier to the cyclocarbonylation process that converts **131c3** to **129c1** was found to be 9.2 kcal/mol. (See the transition state **ts131c3-129c1** in **Figure 48.**) Modeling the Mo-promoted oxidative addition step showed that this step favored the formation of experimentally observed products. The free energy barrier was found to be 14.0 kcal/mol for the formation of the bicyclo [4.3.0] ring system and 9.2 kcal/mol for the formation of the bicyclo [3.3.0] ring system.

Modeling of subsequent steps of the Mo-promoted cyclocarbonylation reaction was easier and more reliable than modeling the analogous Rh-catalyzed steps regardless of the mode of complexation since the oxidative addition step of the molybdenum-promoted reaction always resulted in the formation of either intermediate **129c1** or **132c3**.

4.2.4.2. The carbonyl adsorption step

Next, adsorption of a CO molecule by **129c1** was modeled (**Figure 49**). Placement of carbon monoxide in the vicinity of **129c1** and minimizing the energy of the system does not lead to bond formation between the CO and the molybdenum atom. To get from **129c1** to **159c2** the system has to overcome a barrier (**Figure 50**).



129c1 **159c2**

Figure 49 The carbonyl adsorption step for the Mo-promoted reaction.

This reaction step, shown in **Figure 49**, is an example of why one should consider not only the electronic energy but also the free energy. The electronic energy of the transition state for this step (**ts129c1-159c2**, **Figure 50**) was below that of intermediate **129c1** and a free CO by 2.2 kcal/mol; however, immobilizing a molecule of carbon monoxide decreased the entropy of the system. Therefore, the transition state was actually 8.6 kcal/mol above **129c1** and a CO molecule in free energy. This might seem like a high barrier, but one should remember that the reaction is carried out in toluene. When a CO molecule is immobilized by **129c1**, it probably frees up a solvent molecule, which counteracts unfavorable entropic effects associated with immobilizing a CO molecule. This makes the reaction more entropically favorable, lowering the free energy for the transition state.

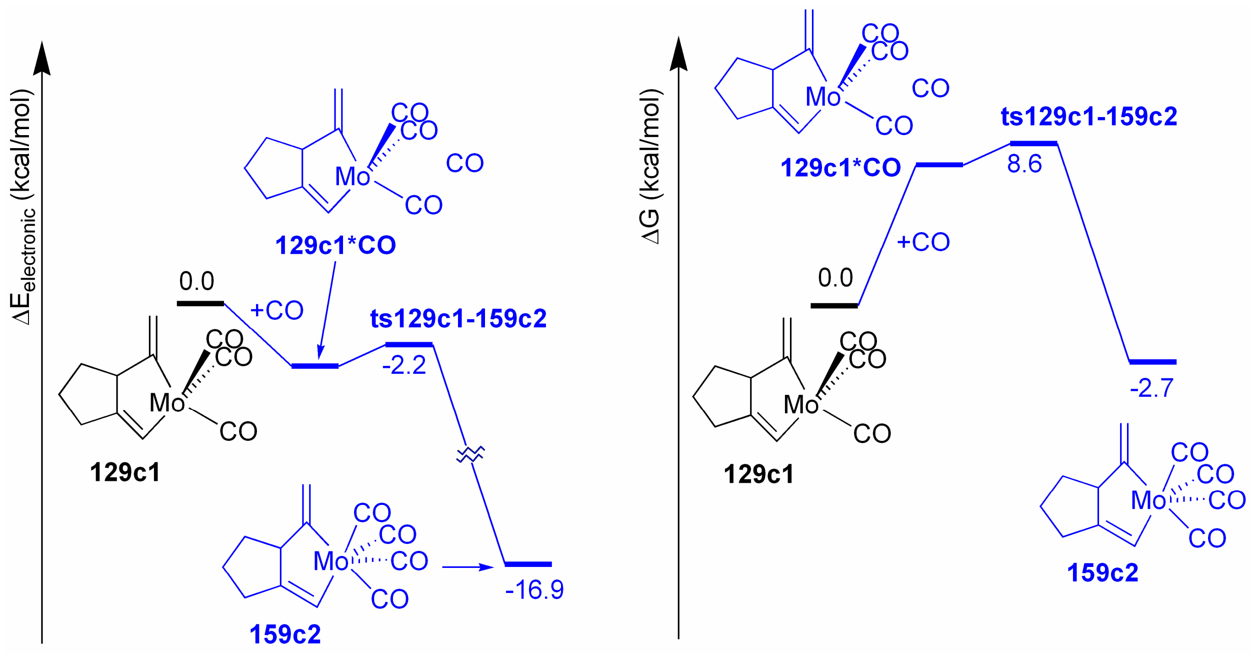


Figure 50 Electronic energy diagram and free energy diagram for the CO adsorption step.

4.2.4.3. The carbonyl insertion step

Carbonyl insertion into metallocycle **159c2** to give **162c1** (**Figure 51**) was calculated to have a free energy barrier of 1.9 kcal/mol.

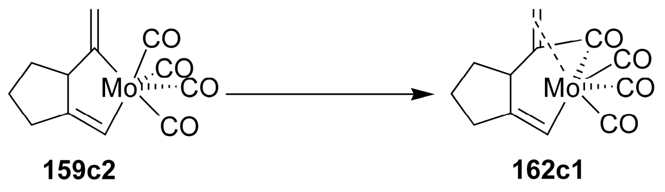
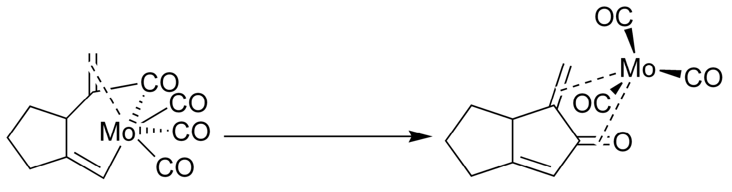


Figure 51 The carbonyl insertion step for the Mo-promoted reaction.

4.2.4.4. The reductive elimination step

Next, the reductive elimination was modeled. The transition state between **162c1** and the product of this step was found using coordinate-driven methods. The free energy barrier for this process was found to be 1.8 kcal/mol. The reductive elimination of molybdenum afforded the complexed enone **163c1**, representing a free energy change of -42.1 kcal/mol.



162c1 163c1

Figure 52 The reductive elimination step for the Mo-promoted reaction.

4.2.4.5. Modeling alternative mechanisms for the Mo-promoted reaction

There could be alternative mechanisms to the Mo-promoted reaction differing from the one described above. Appendix B discusses an alternative way for CO ligation of metallocycle **129c1**. Appendix C discusses the reaction mechanism where the ligation of molybdenum by a carbon monoxide molecule from the media is not the next step after the oxidative addition step.

4.2.4.6. Conclusions about the Mo-promoted intramolecular cyclocarbonylation

The free energy landscape for the Mo-promoted intramolecular cyclocarbonylation is shown in **Figure 53**. Although the rate limiting step turned out to be the addition of a carbon monoxide molecule to the molybdenum center, the product was determined by the oxidative addition step. In the reaction path that leads to the formation of the [4.3.0] bicyclic ring system, the transition state for this step was 4.8 kcal/mol above that for the oxidative addition step and 2.5 kcal/mol above that for the CO adsorption step of the reaction path that leads to the formation of the [3.3.0] bicyclic ring system. Further steps, carbonyl insertion and reductive elimination, encountered small free energy barriers.

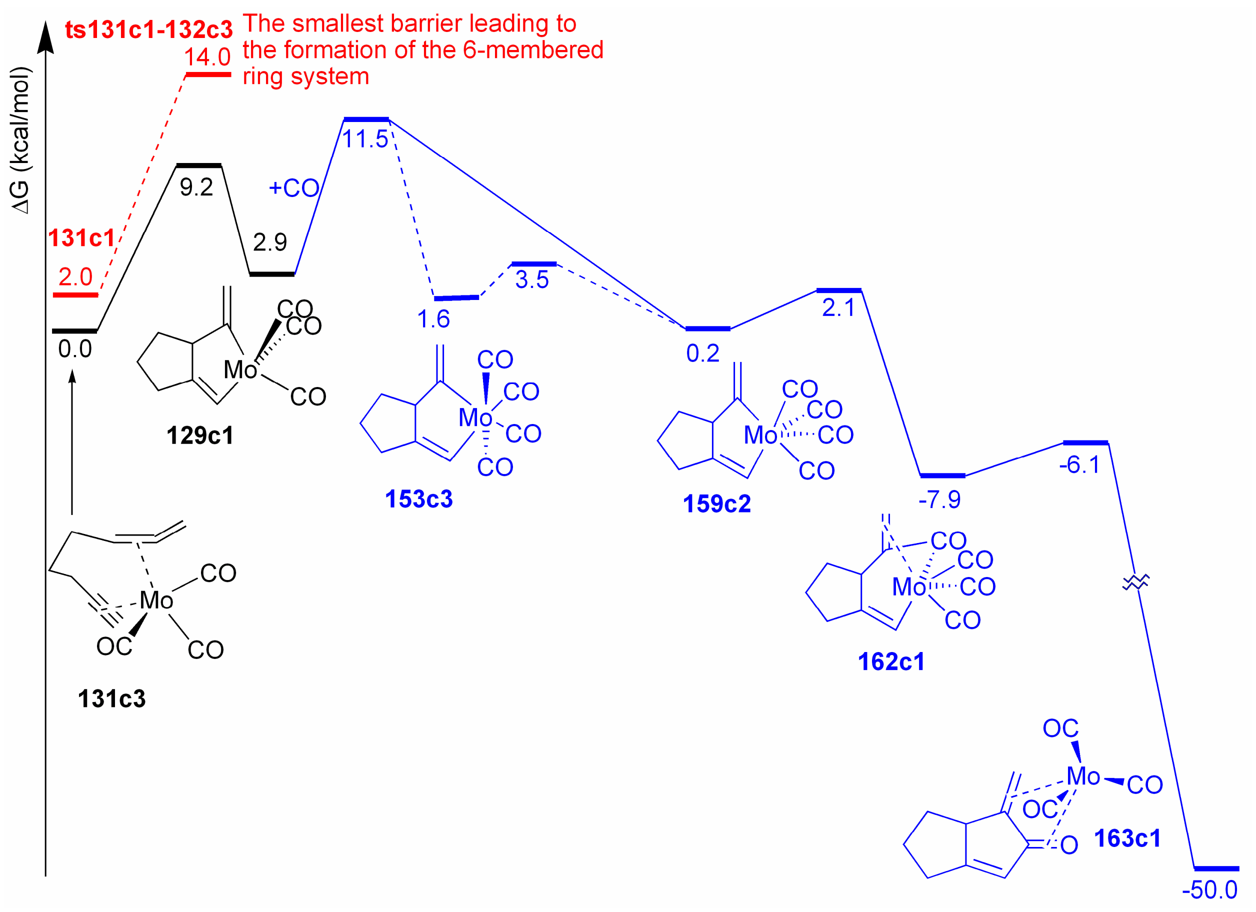


Figure 53 Mo-promoted cyclocarbonylation mechanisms, where CO adds to the Mo center immediately following the ring formation.

Parts of pathways with an extra CO present are shown in blue. The best possible pathways found in this study are shown in solid lines. Other pathways are shown in dashed lines.

4.2.5. Modeling rhodium-catalyzed bicyclic ring formation

4.2.5.1. The oxidative addition step

For the rhodium-catalyzed intramolecular [2 + 2 + 1] cyclocarbonylation reaction the oxidative addition step was found to require overcoming the highest transition state, so it was assumed at the outset that this step was the most important one for the analogous intermolecular process, and it was modeled first. Modeling this step produced information such as the structure of intermediate **127c6**. This information was used in the modeling of subsequent steps.

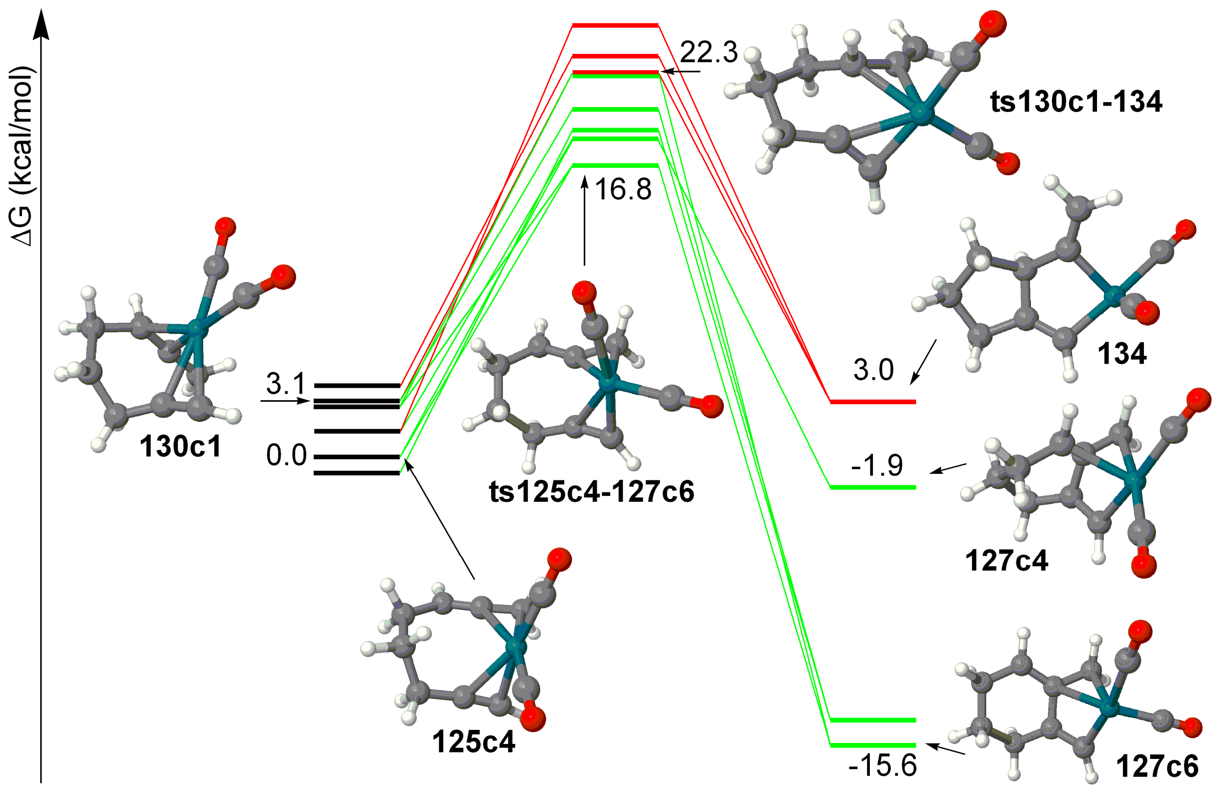


Figure 54 Energetics of the Rh-catalyzed oxidative addition step.

Pathways that led to the experimentally observed product are shown in green and those that led to the products that are not formed in the experiment are shown in red.

Complexation of rhodium biscarbonyl to the alkyne and the distal bond of the allene was energetically favored by 3.1 kcal/mol over the competing complexation product where the rhodium was coordinated to the proximal double bond of the allene, **130c1**. Other low-energy structures were found but they only differed by conformations of the carbon tether.

Starting with various conformations of compounds **130** and **125** and using coordinate-driven methods resulted in the formation of compounds **134** and **127** (see **Figure 54**). Additional transition states and reactant isomers were identified for the carbocyclization step by using coordinate-driven methods, but starting with **134** and **127** and searching for reactants. We also searched for transition states by modifying low-energy transition states found for the molybdenum-promoted oxidative addition step. The transition state between **125c6** and **134** was found this way.

All low-energy transition states (**Figure 54**) showed the rhodium biscarbonyl most closely associated with the distal double bond of the allene. Just as for the Mo-promoted carbocyclization step reactants, all low-energy structures only differed by conformational changes in the carbocyclic tether. Large energy differences were observed for metallocycles **127c6**, **127c4** and **134**. Clearly the bicyclo[4.3.0] ring system found in **127c6**, **127c4** was much more stable than the competing bicyclo[3.3.0] ring system found in **134**.

These calculations were consistent with the experimentally observed products. The unexpected result was the prediction that the oxidative addition step of the rhodium catalyzed reactions had to overcome larger barriers than the analogous step of the molybdenum catalyzed reactions. The Gibbs free energy barrier for the formation of the [4.3.0] ring system was calculated to be 16.8 kcal/mol for the rhodium-catalyzed reaction, which is close to 14.0 kcal/mol, the barrier height for the analogous molybdenum-promoted reaction. As for the [3.3.0] ring formation, the lowest free energy barrier for the rhodium-catalyzed reaction was found to be 22.3 kcal/mol, much higher than the 9.2 kcal/mol free energy barrier for the analogous molybdenum-promoted reaction.

4.2.5.2. The carbonyl adsorption step

In the molybdenum-promoted reaction described above, the adsorption of an extra CO by the metal atom (**Figure 49**) required overcoming a 8.6 kcal/mol free energy barrier. In the rhodium-catalyzed reaction the energetics of this step were different. The attachment of the carbonyl group to **127c6** was found to proceed without a barrier, resulting in the formation of metallocycle **148c1** (**Figure 55**). In this process the free energy decreased by 2.5 kcal/mol.

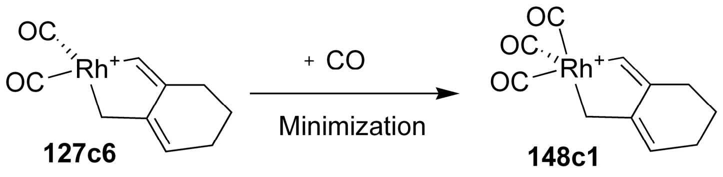


Figure 55 The carbonyl adsorption step for the Rh-catalyzed reaction.

4.2.5.3. The carbonyl insertion step

Carbonyl insertion into metallocycle **148c1** to give **149c2** (**Figure 56**) was calculated to have a free energy barrier of 1.0 kcal/mol. Intermediate **149c2** was 14.8 kcal/mol lower in free energy than metallocycle **148c1**.

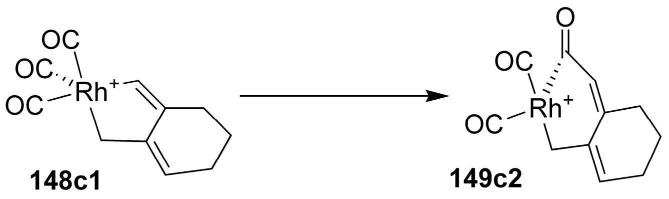


Figure 56 The carbonyl insertion step for the Rh-catalyzed reaction.

4.2.5.4. The reductive elimination step.

Reductive elimination of the rhodium center from **149c2** (**Figure 57**) required overcoming an 11.5 kcal/mol free energy barrier. This is much higher than for the analogous Mo-promoted reaction shown in **Figure 52**. The free energy drop in this reaction step, 14.9 kcal/mol, is also not as large as that for the Mo-promoted reaction. In intermediate **147c1** formed in this step, the metal center is attached to the ring system somewhat differently than in the analogous molybdenum-containing intermediate **163c1** shown in **Figure 52**. In **147c1** it is attached to two alkene groups by η^2 bonds. In **163c1** molybdenum is attached to one alkene and one carbonyl group in a similar fashion.

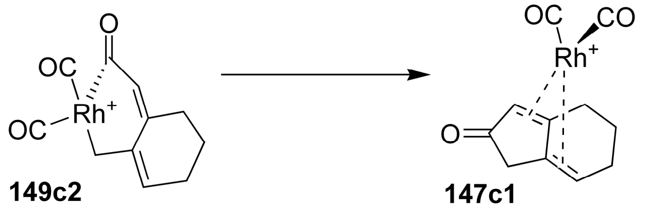


Figure 57 Reductive elimination step for the Rh-catalyzed reaction.

4.2.5.5. Modeling alternative mechanisms for the Rh-catalyzed cyclocarbonylation reaction

There could be alternative mechanisms for the Rh-catalyzed [2 + 2 +1] intramolecular cyclocarbonylation reaction differing from the one described above. Appendix E discusses the reaction mechanisms where there is no CO adsorption immediately following the oxidative addition step. Appendix D explores the oxidative addition step with three carbonyl ligands on the rhodium atom. Appendix F discusses our attempts to model a mechanism where CO inserts into the allene or alkyne, prior to the oxidative addition step.

4.2.5.6. Conclusions about Rh-catalyzed intramolecular cyclocarbonylation

The energetics of the rhodium-catalyzed [2 + 2 + 1] cyclocarbonylation reaction, shown in **Figure 58**, are much more like energetics of the rhodium-catalyzed intermolecular cyclocarbonylation reaction modeled earlier than the molybdenum-promoted intramolecular cyclization reaction. It was assumed that the complexation of $Rh(CO)_2^+$ to the unsaturated hydrocarbon molecule does not require overcoming a significant barrier. The step that follows, oxidative addition, determines both the rate and the product of the reaction. After oxidative addition a carbon monoxide is added to the rhodium center from the media. Unlike the analogous molybdenum-promoted reaction, which requires overcoming a significant energy barrier, in the case of the rhodium-catalyzed reaction this step proceeds without a barrier. The

CO insertion step requires overcoming only a very small barrier. The step that follows, the elimination of rhodium from the ring system, requires overcoming a significant barrier; however, it is not as large as the one for the oxidative addition step.

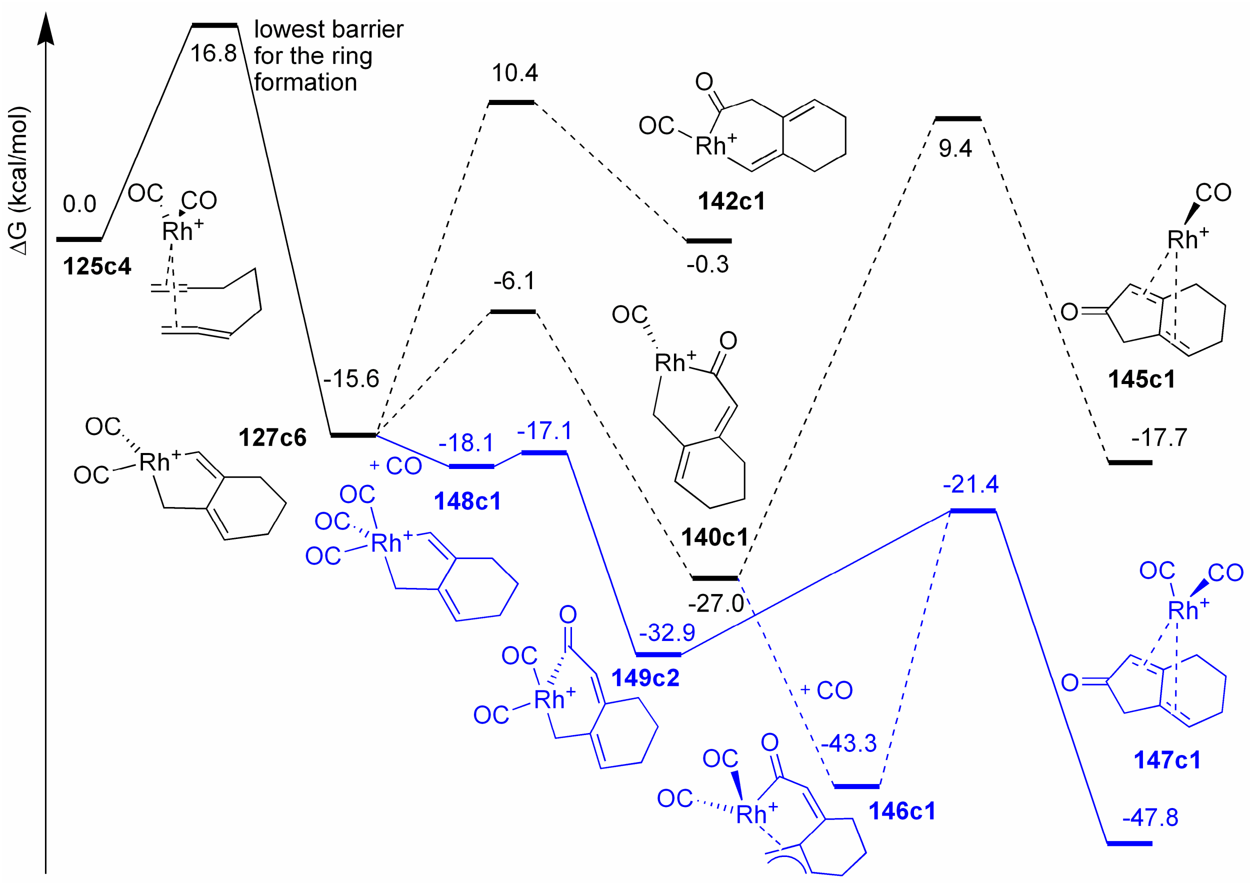


Figure 58 Energetics of the Rh-catalyzed cyclocarbonylation.

Parts of pathways with an extra CO present are shown in blue. The lowest energy pathway found in this study is shown in solid lines. Other pathways are shown in dashed lines.

4.3. Conclusions

4.3.1. A comparison between the Rh-catalyzed and the Mo-promoted [2 + 2 + 1] cyclocarbonylation reactions of allenes.

We found that in both the rhodium-catalyzed and the molybdenum-promoted [2 + 2 + 1] cycloisomerization reactions of allenes the transition states for the oxidative addition step were high in free energy. For the rhodium-catalyzed reaction the transition state for the oxidative addition step was the highest in free energy, and for the molybdenum-promoted reaction it was the second highest. In the case of the rhodium-catalyzed reaction, the oxidative addition step determined both the rate and the product of the reaction. The situation for the molybdenum-promoted reaction was slightly different. Although the oxidative addition step determined the product, the rate of this reaction was controlled by the next step, the attachment of a carbon monoxide molecule from the media to the molybdenum atom.

The geometries of transition states for the oxidative addition step were different for the rhodium-catalyzed and the molybdenum-promoted reactions. In low-lying transition states the molybdenum atom was bound to the proximal double bond of the allene and was in the plane of the hydrocarbon ring. Rhodium was bound to the distal double bond of the allene and was outside the plane of the hydrocarbon ring. Unlike the molybdenum-containing transition states, the rhodium-containing transition states for the oxidative addition step were early transition states. They were more reminiscent of reactants than of products both in geometry and energy.

The free energy barrier for the oxidative addition that results in the [4.3.0] bicyclic ring system formation was about the same for both the Rh-catalyzed and the Mo-promoted reactions, 16.8 kcal/mol and 14.0 kcal/mol respectively. However, the barriers for the [3.3.0] bicyclic ring system formation were very different, 9.2 kcal/mol for the molybdenum-promoted reaction and 22.3 kcal/mol for the rhodium-catalyzed reaction. This means that the rhodium catalysis produces a [4.3.0] bicyclic ring system not because it is efficient at this reaction, but rather because it is inefficient at producing the competing [3.3.0] bicyclic ring product.

The [4.3.0] bicyclic ring system is more stable because in it the two alkene groups are conjugated with each other. The existence of low-lying transition states for the oxidative addition and carbon monoxide adsorption steps of the Mo-promoted reaction leads to the formation of a thermodynamically unfavored [3.3.0] bicyclic ring system. This is an example of a kinetically controlled reaction.

The steps that followed the addition of a carbon monoxide molecule from the media to the metal atom – insertion of a CO molecule into the metallocycle and reductive elimination of metal from the ring system – did not require overcoming significant barriers. For both metal catalysts the product is determined early in the reaction.

4.3.2. Predictions

4.3.2.1. Improving the Mo-promoted reaction

In the case of the molybdenum-promoted reaction, the oxidative addition step is not irreversible. The step that follows, the addition of a carbon monoxide molecule from the media to the molybdenum center, requires overcoming a larger barrier than the oxidative addition step. The height of the barrier for the addition of a carbon monoxide molecule from the media to the molybdenum center has two consequences:

1) The reaction rate is lower.

2) Because the reaction rate is lower, other reactions have more time to compete with the formation of desired products, which decreases the yield.

If one increases the pressure of carbon monoxide, the reaction rate and yield should both increase.

4.3.2.2. Improving the Rh-catalyzed reaction

Rhodium- and iridium-based catalysts are expensive. It is preferable to replace these with a catalyst based on a more accessible metal, such as cobalt. However, there is a problem with cobalt-based catalysts – they tend to form Co-Co bonds and catalyze formation of rings of varying sizes.

One might consider putting a bulky tridentate ligand on cobalt(I) and using that complex as a catalyst for reactions described in this chapter. If a complex like this dimerizes by forming Co-Co bonds, the cobalt atoms will become shielded from the environment by the bulky tridentate ligands, so they will not be able to participate in catalysis and will not lead to the formation of undesired products. The monomer of this complex will insert itself into the ring more like rhodium than like molybdenum, because the bulky tridentate ligand will prevent the cobalt atom from being in the plane of the hydrocarbon ring, coordinated to the proximal bond of the allene like the molybdenum atom.

5. Rhodium(I)-catalyzed allenic Pauson-Khand type reaction and mechanistic studies using theoretical methods.

5.1. Introduction

Professor Brummond's group has explored several reactions related to the one described in the previous chapter. Because Rh(I)/Ir(I)-catalyzed allenic Alderene reactions lead to the formation of cyclized cross-conjugated trienes in high yields and take only minutes, Professor Brummond's group was interested in testing a related reaction, where intramolecular condensation of allene and alkene leads to the formation of heterocyclic rings.

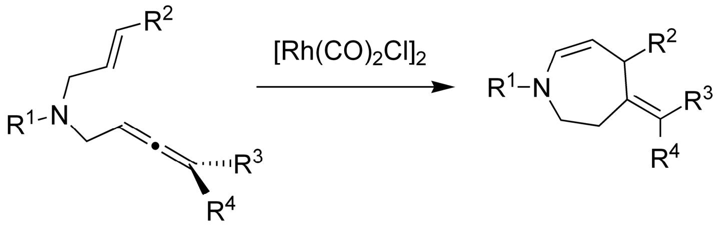


Figure 59 Intramolecular condensation reactions observed by Prof. Brummond's group. R^1 is an electron withdrawing group. The reaction works best in polar aprotic solvents.

The use of alkene instead of alkyne leads to the formation of unexpected products.[100] (**Figure 59**) These novel reactions can be useful in the synthesis of natural products and natural product-like compounds.[101, 102]

All experiments described in this chapter were carried out in Prof. Brummond's group unless noted otherwise.

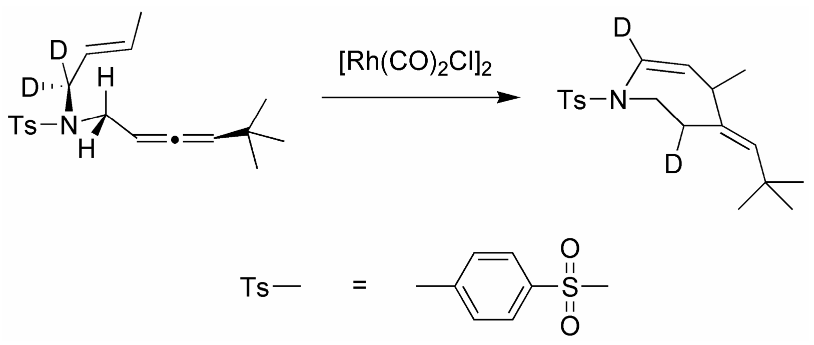


Figure 60 A deuterium labeling study of Rh(I)-catalyzed azepine formation. One hydrogen atom transfers to the other side of the ring.

The mechanism of these reactions is not obvious. To gain insight into the mechanism a deuterium labeling study was done (**Figure 60**). It showed that in the course of the reaction a hydrogen atom transfers to the other side of the ring. However, some questions remain about the mechanism of this reaction. One has to do with the sequence of reaction steps. Did the hydrogen atom transfer before or after the ring was formed? Another question concerns reaction specificity. In principle, the hydrogen from the other side of the ring could transfer across the ring leading to the formation of more stable products (**Figure 61**). However, this reaction did not occur. This was puzzling.

To answer these questions various steps of this reaction were modeled using DFT.

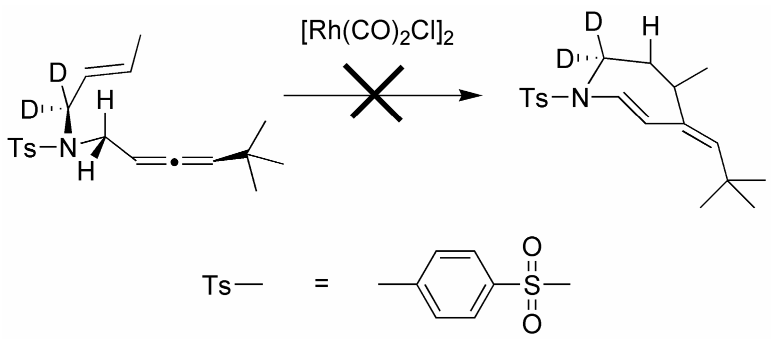


Figure 61 The reaction that was not observed.

In the product of this reaction the C-C double bonds are conjugated with each other, so it should be more stable than the experimentally observed product shown in Figure 60.

5.2. Experimental

5.2.1. Basis Set/Level of theory

All calculations were performed using GAUSSIAN98[103] and GAUSSIAN03[92] using density functional theory with Becke's three-parameter hybrid exchange functional and the Lee-Yang-Parr correlation functional (B3LYP).[20] The 6-31G(d) basis set[93] for H, C, N, O and F atoms and the double-ζ quality, Hay and Wadt LANL2DZ basis set was used in conjunction with the LANL2 effective core potential[94] for Rh and Cl. An additional d polarization function with coefficient 0.6 was added to the basis set for chlorine atoms. Transition states were found using the QST3, QST2, and coordinate-driven methods.

For simplicity the electron withdrawing group was modeled by a fluorine atom in all calculations.

5.2.2. Solvation model

All stationary points were optimized in a vacuum, and then the energies of stationary points were corrected for solvation in boiling THF using COSMO.[21]

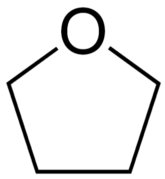


Figure 62 Tetrahydrofuran (THF)

The dielectric constant used for THF was 5.44, a number found by a least squares linear fit of the dielectric constant of THF at 25 °C, 30 °C and 50 °C.[104]

5.3. Discussion and Results

5.3.1. Hydride transfer mechanism.

We started with compound **59** and tried to move the appropriate hydrogen atoms to rhodium. We obtained a surprising result. Instead of forming compounds **80** and **120**, **59** transformed into intermediates **86** and **89** (**Figure 63**). This came as a surprise, because we did not expect the positive charge to be transferred form rhodium to nitrogen. We have tried modeling this reaction step backwards starting with **80** and **120**, but **120** minimized to **89**, and **80** to **96**.

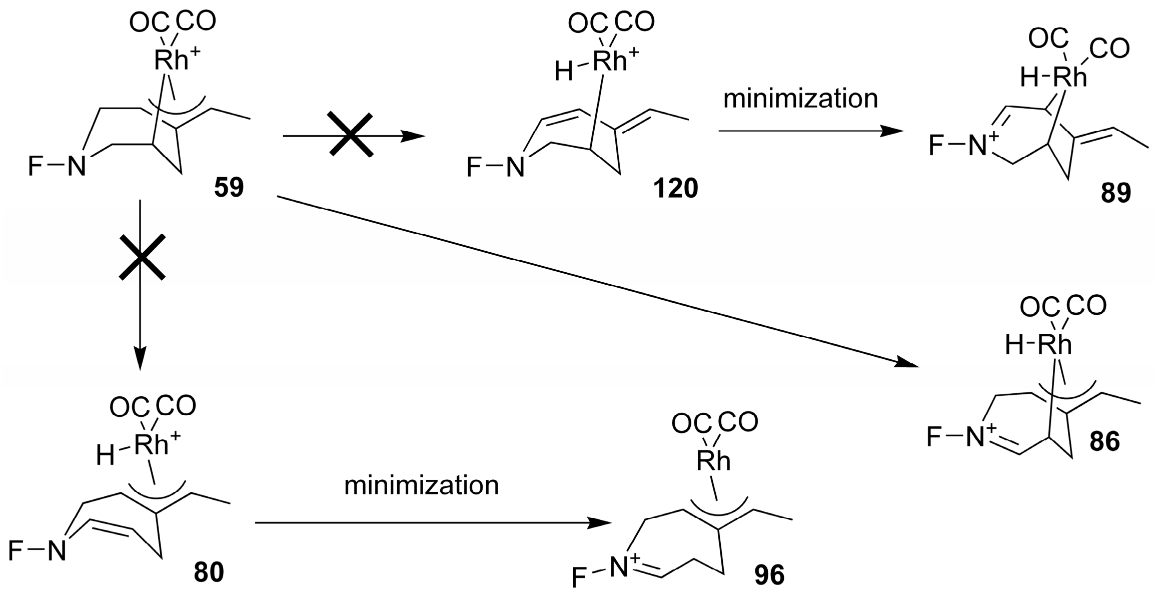


Figure 63 Unexpected hydride transfer reactions.

When we started with compound 59 and tried to move the appropriate hydrogen atom to rhodium, 59 transformed into intermediates 86 and 89 instead of 80 and 120.

We modeled the second step of the reaction by moving the rhodium-attached hydrogen atoms in compounds **86** and **89** to the appropriate carbon atoms. This lead to the formation of compounds **88** and **87**. We have also tried to accomplish these reactions in one step by moving the hydrogen atoms directly to the appropriate carbon atoms. These calculations went through only one transition state, corresponding to the transition state of the second step. We found the second step was the rate determining step in this process. The transition state on the path, that leads to experimentally observed products, had Gibbs free energies of 2.0 kcal/mol and 6.9 kcal/mol relative to compound **59**, while the transition states between compounds **59** and **89** and compounds **89** and **87** had Gibbs free energies of 9.3 kcal/mol and 10.5 kcal/mol, so the correct product was predicted.

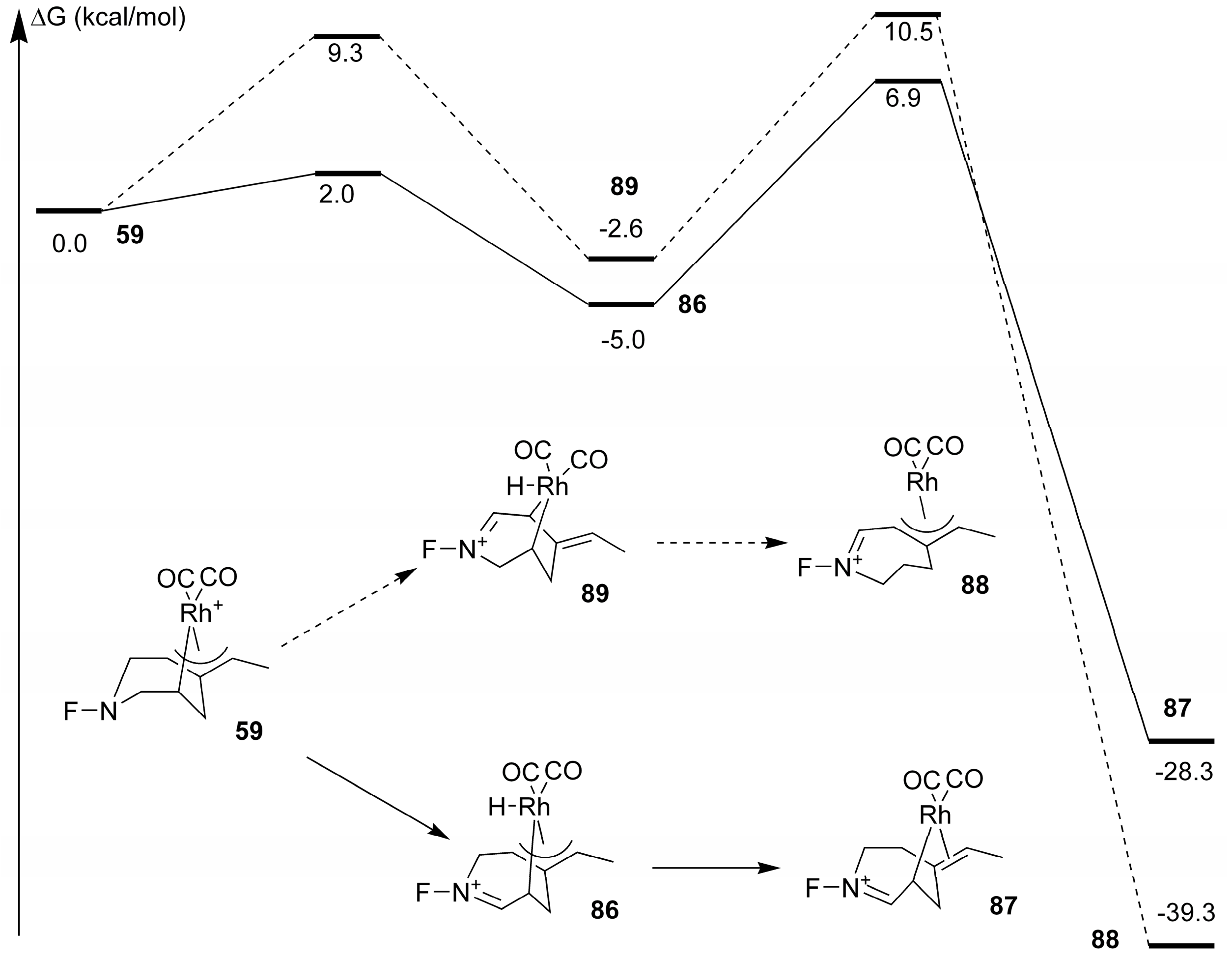


Figure 64 Hydride transfer mechanism involving two CO ligands.

Solid lines and arrows indicate the reaction that leads to analogs of experimentally observed products.

5.3.2. Modeling a reaction that can only proceed by β-hydride elimination

The hydride transfer mechanism was unexpected; we had expected the reaction to proceed via β-hydride elimination. This led us to study a related system in which the reaction cannot proceed by hydride transfer. To do that, we replaced the nitrogen atom in our models with a carbon atom with an attached fluorine atom. The process we modeled was similar to reactions observed by Makino and Itoh (**Figure 65**).[105]

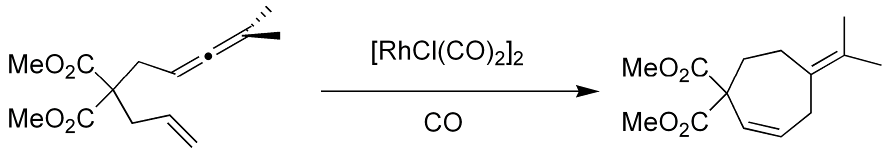


Figure 65 One of reactions observed by Makino and Itoh.

The preferred pathway again led to products similar to those observed experimentally. However, there was a bigger energy gap between the starting compound and the transition states: 10.8 kcal/mol and 18.0 kcal/mol for the pathway that led to the experimentally observed product analogs and 19.1 kcal/mol and 24.0 kcal/mol for the one that did not.

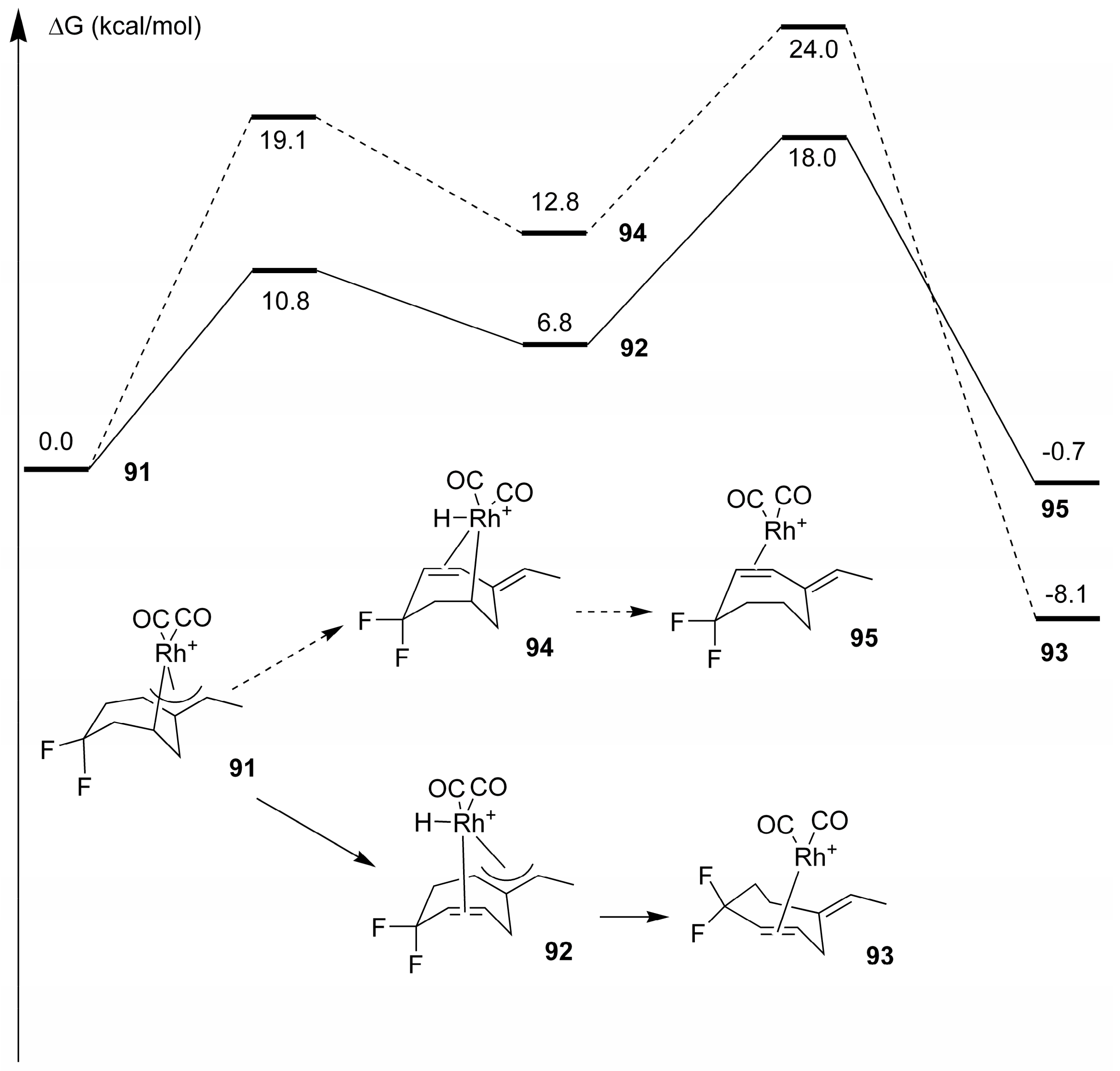


Figure 66 β-Hydride elimination pathways for reactions similar to reactions observed by Makino and Itoh. Solid lines and arrows indicate the reaction that leads to analogs of experimentally observed products.

5.3.3. A β-hydride elimination reaction involving a chloride ligand.

We thought that chloride might be one of the ligands attached to the rhodium in the reaction we were trying to model. Modeling a reaction where a chloride is attached to rhodium is more involved, because swapping a chloride and the carbonyl results in another isomer, so one has to model two sets of pathways depending on which side the chloride occupies. Unlike the pathways beginning with compounds **59** and **91**, we decided to start the modeling of chloride-containing pathways with the intermediates, because the intermediates were easier to guess than the rhodium-containing compounds in the beginning of simulated reaction pathways. We started with the geometries of compounds **86** and **89** and replaced one of the carbonyl groups in them with a chloride, making each of them into two isomers of **102** and **104**. These had somewhat different geometries from the square planar complexes **86** and **89**, so we searched for different conformers, but did not find any other isomers relevant to the reaction.

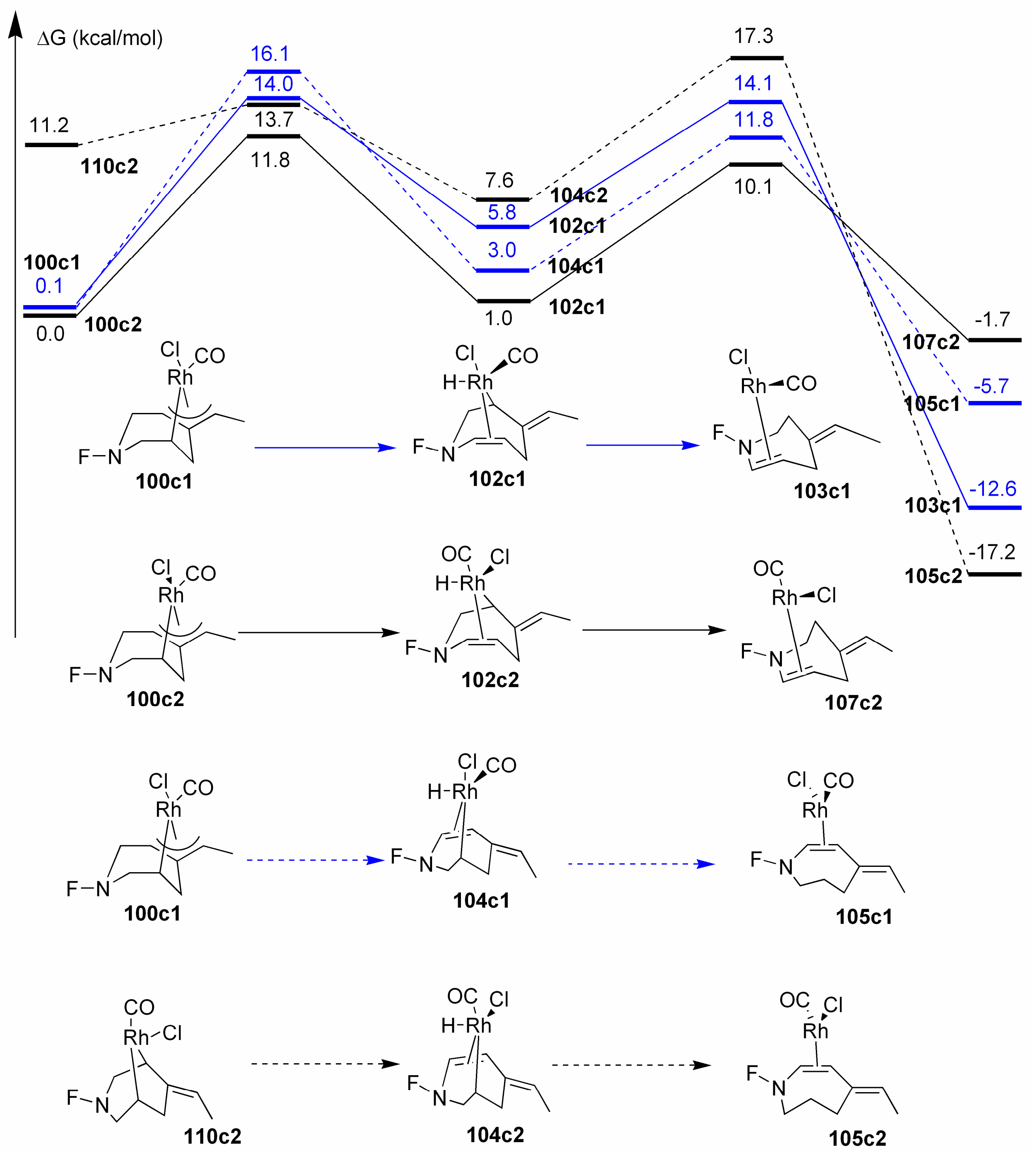


Figure 67 β-Hydride elimination pathways involving chlorine.

Solid lines and arrows indicate reactions that lead to analogs of experimentally observed products. Some parts of the pathway are shown in grey and others in black. The difference between these are the positions of chloride and carbonyl ligands.

Starting with the relevant isomers of **102** and **104** we have traced the reaction pathways both forward and backward using the methods described earlier (**Figure 67**). Again, the modeling favored a pathway that leads to the analogs of the observed products. However this time the free energy gap was greater between the staring compounds and the transition states than in the analogous pathway that had two carbonyls attached to the rhodium atom: 11.8 kcal/mol and 10.1 kcal/mol for one pathway that leads to the observed product analogs and 14.0 kcal/mol and 14.1 kcal/mol for the other. The transition states for the other pathways were even higher in free energy: 13.8 kcal/mol and 17.3 kcal/mol for one pathway and 16.1 kcal/mol and 11.8 kcal/mol for the other. The first free energy barrier in these pathways is comparable in size to the second barrier. This pattern is different from the pathways where two carbonyl groups are bound to rhodium, where the first barrier is much lower in free energy than the second one.

5.3.4. A comparison of β-hydride elimination and hydride transfer

5.3.4.1. A calculation favoring β-hydride elimination

To aid in understanding whether a covalently bound chloride is attached to the rhodium atom in the course of the reaction, we compared the free energies of the highest transition states on the preferred pathways. The two - carbonyl pathway was higher in free energy by 25.3 kcal/mol in the tightly bound chloride model and by 43.0 kcal/mol in the infinite dilution model. These results are not very reliable, though, because the calculation is very sensitive to electrostatic effects and small changes in solvent properties can easily make the two-carbonyl pathway lower in energy by better accommodating the chloride ion.

5.3.4.2. Experimental evidence favoring β-hydride elimination

A piece of experimental evidence that supports β-hydride elimination mechanism is the fact that the reaction proceeds better when there is an electron-withdrawing group on the nitrogen atom. An electron-donating group would have speeded up the reaction if it proceeded by the hydride transfer mechanism.

5.3.4.3. Solvent effects – inconclusive experimental evidence

The solvent effects can be used be used to make arguments in favor of both the hydride transfer mechanism and the β-hydride elimination. On the one hand, our calculations did not show that the presence of the solvent significantly decreases the energy gap between the starting materials and the transition states in our models, but these reactions tend to proceed better in polar aprotic solvents. The role of these solvents could be to keep chloride dissolved and away from rhodium allowing the two-carbonyl hydride transfer reaction to take place.

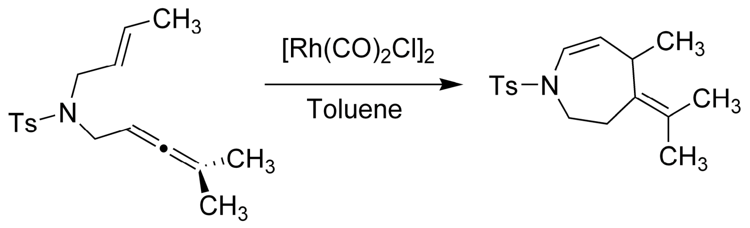


Figure 68 This reaction occurred in a non-polar solvent.

On the other hand, there was one instance where the reaction was observed in toluene (**Figure 68**). It is clear that in this case the chloride could not stay in the solution as a free ion, so the reaction probably proceeded by β-hydride elimination.

5.3.4.4. Introduction of Ag^+ into the reaction mixture - inconclusive experimental evidence

One way to determine if chloride is critical to the catalytic pathway is to eliminate it from solution by precipitating it with silver cations. This can be accomplished by adding a soluble silver salt to the reaction mixture. (See **Figure 37** on page 63.) The ability of the reaction to proceed with silver cations in solution would be an indication that chloride is not present in the catalyst. This would have been strong evidence for the hydride transfer reaction.

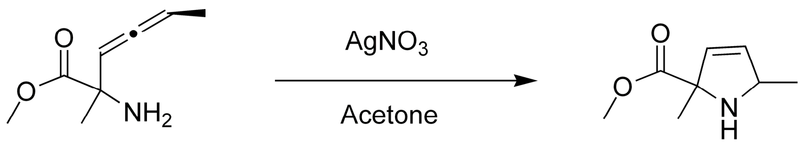


Figure 69 A silver-catalyzed reaction recently discovered by Prof. Brummond's group. Similar processes can compete with the rhodium-catalyzed reaction discussed in this chapter.

The results of experiments where silver salts were introduced into the reaction mixture were inconclusive. Although the reaction did not produce the desired products, this does not prove that the presence of chloride is vital, because silver cations could participate in competing reactions. An example of such reaction is shown in **Figure 69**.

5.3.5. β-Hydride elimination before cyclization

The unlikely possibility that β-hydride elimination occurs before cyclization was also considered. Modeling was started with intermediate **167c1** in which the tether has not undergone cyclization yet. The η^2 bond between rhodium and allene can break in **167c1** and the hydrogen atom bound to C3 can transfer onto rhodium which leads to the formation of intermediate **166c1** (**Figure 70**). Because the free energy barrier this process was high, 30.1 kcal/mol, solvent effects were not considered.

An important feature of intermediate **166c1** was that the rhodium-bound hydrogen was located on one side of the rhodium atom and the allene was located on the opposite side. This meant that even if this reaction step occurred successfully, it could not be followed by the transfer of hydrogen to the allene.

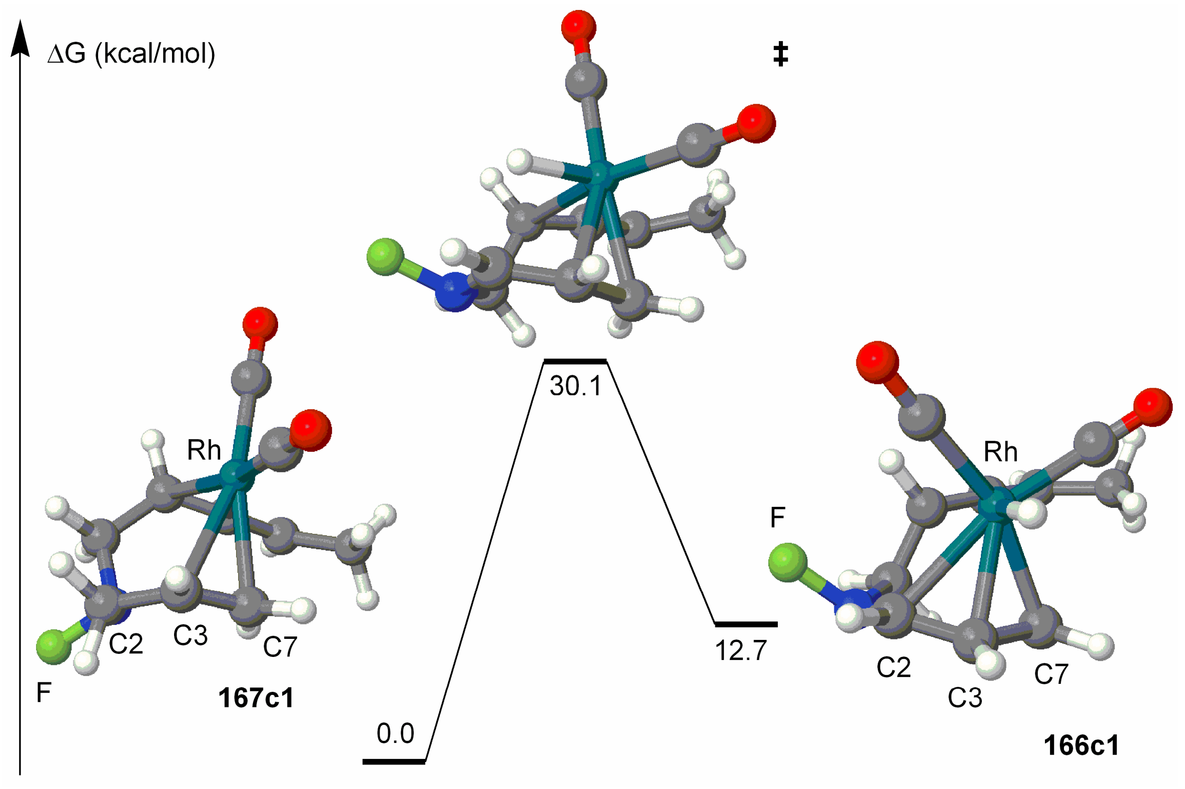


Figure 70 β-Hydride elimination prior to carbocyclization.

5.4. Conclusions

Our calculations showed that the transfer of hydrogen from one side of the ring to another occurs after the cyclization step. The hydrogen transfers from the alkene and not from the allene side of the ring for two reasons:

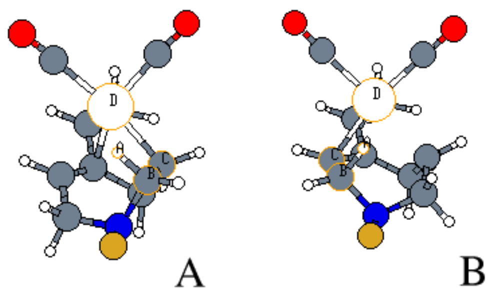


Figure 71 Conformations that 59 must adopt to start hydride transfer.

Conformation A leads to the experimentally observed product. It takes 5.5 kcal/mol to get into that conformation. Conformation B does not lead to the experimentally observed products. It takes 20.4 kcal/mol to get into this conformation.

1) The ring is more flexible on the alkene side, so it is easier for the molecule to adopt a conformation where the dihedral angle between the alkene hydrogen and rhodium allows for hydride transfer or β-hydride elimination (**Figure 71**).
2) In transition states that lead to the placement of the transferred hydrogen into the position on the other side of the ring, the rhodium atom is stabilized by C-C double bonds. If the hydrogen is transferred to the allene side of the ring, the transition states can be stabilized by two C-C double bonds, while if it is transferred to the alkene size of the ring it can only be stabilized by one.

Although there is both experimental and theoretical evidence that these reactions occur through β-hydride elimination (**Figure 67**), the calculations raised the possibility that it might actually go through hydride transfer (**Figure 64**). The free energy barrier for the β-hydride elimination step is low, only 11.8 kcal/mol, so the rate-determining step in rhodium(I) catalyzed allenic Pauson-Khand type reactions is probably carbocyclization. The proposed reaction mechanism is shown in **Figure 72**.

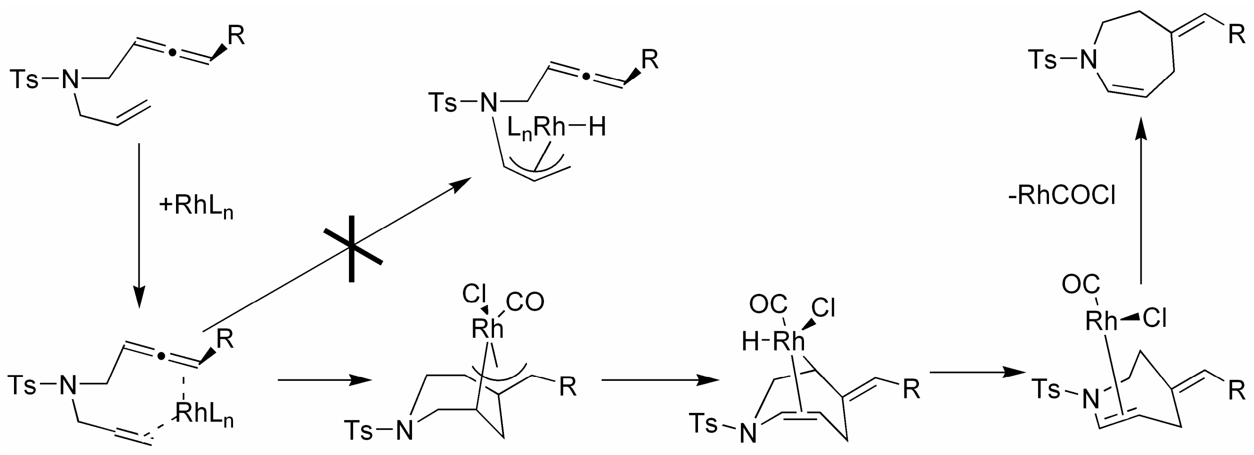


Figure 72 The proposed reaction mechanism.

The hydrogen atom transfers to the other side of the heterocycle by a process that involves β-hydride elimination after carbocyclization.

Table 1 The computed energies of minima discussed in the text.

Minimum Name	$E_{el\ vac}$ (au)	$E_{el\ THF}$ (au)	ZPE $_{THF}$ (au)	E $_{THF}$ (au)	H $_{THF}$ (au)	G $_{THF}$ (au)
59	-802.518624	-802.581136	-802.375786	-802.356901	-802.355827	-802.424886
86	-802.521412	-802.582953	-802.380239	-802.360024	-802.358950	-802.432903
89	-802.518807	-802.579626	-802.376513	-802.356361	-802.355287	-802.428988
96	-802.553439	-802.622103	-802.415657	-802.395393	-802.394319	-802.469281
88	-802.578678	-802.641048	-802.433563	-802.413224	-802.412150	-802.487497
87	-802.561750	-802.623696	-802.416465	-802.396224	-802.395150	-802.469908
91	-885.800285	-885.864884	-885.654846	-885.633928	-885.632854	-885.708059
92	-885.786635	-885.850207	-885.643323	-885.622180	-885.621106	-885.697191
93	-885.799943	-885.862721	-885.653246	-885.631798	-885.630724	-885.709170
94	-885.777092	-885.840409	-885.633217	-885.611995	-885.610921	-885.687616
95	-885.813403	-885.875924	-885.665663	-885.644352	-885.643278	-885.720997
100c1	-704.405974	-704.422778	-704.223495	-704.204760	-704.203686	-704.274415
102c1	-704.392573	-704.407342	-704.212327	-704.192847	-704.191773	-704.265219
103c1	-704.423166	-704.438675	-704.239892	-704.220351	-704.219277	-704.294651
100c2	-704.407286	-704.422330	-704.223209	-704.204303	-704.203229	-704.274523
102c2	-704.397289	-704.414670	-704.219558	-704.199969	-704.198895	-704.273003
107c2	-704.405708	-704.420145	-704.222535	-704.202665	-704.201591	-704.277200
104c1	-704.396043	-704.412554	-704.217027	-704.197588	-704.196514	-704.269813
105c1	-704.413611	-704.427949	-704.229693	-704.209914	-704.208840	-704.283644
110c2	-704.389029	-704.401493	-704.204604	-704.185555	-704.184481	-704.256654
104c2	-704.390495	-704.403720	-704.209102	-704.189407	-704.188333	-704.262373
105c2	-704.431659	-704.447010	-704.247925	-704.228310	-704.227236	-704.301897

Minimum Name	$E_{el\ vac}$ (au)	ZPE $_{vac}$ (au)	E $_{vac}$ (au)	H $_{vac}$ (au)	G $_{vac}$ (au)
167c1	-802.504649	-802.301955	-802.280170	-802.279096	-802.357387
166c1	-802.481897	-802.281906	-802.260164	-802.259090	-802.337160

$E_{el\ vac}$, ZPE $_{vac}$, E $_{vac}$, H $_{vac}$, G $_{vac}$, $E_{el\ THF}$, ZPE $_{THF}$, E $_{THF}$, H $_{THF}$ and G $_{THF}$ stand for electronic energy in vacuum, zero point energy in vacuum, energy in vacuum, enthalpy in vacuum, free energy in vacuum, electronic energy in THF, zero point energy in THF, energy in THF, enthalpy in THF and free energy in THF respectively.

Table 2 The computed energies of transition states discussed in the text.

TS name	$E_{el\ vac}$ (au)	$E_{el\ THF}$ (au)	ZPE $_{THF}$ (au)	E $_{THF}$ (au)	H $_{THF}$ (au)	G $_{THF}$ (au)
Hydride transfer mechanism involving two CO ligands.						
ts59-86	-802.507737	-802.571323	-802.370254	-802.350791	-802.349717	-802.421664
ts86-87	-802.500312	-802.561695	-802.360500	-802.340427	-802.339353	-802.413832
ts59-89	-802.494752	-802.557971	-802.357357	-802.337583	-802.336509	-802.410107
ts89-88	-802.494398	-802.556262	-802.354737	-802.334642	-802.333568	-802.408152
β-Hydride elimination pathways for reactions similar to reactions observed by Makino and Itoh.						
ts91-92	-885.779694	-885.843634	-885.638014	-885.617471	-885.616397	-885.690861
ts92-93	-885.766853	-885.830513	-885.624944	-885.603960	-885.602886	-885.679363
ts91-94	-885.765601	-885.828580	-885.623396	-885.602466	-885.601392	-885.677546
ts94-95	-885.757319	-885.820424	-885.614884	-885.593751	-885.592677	-885.669764
β-Hydride elimination pathways involving chloride.						
ts100c1-102c1	-704.376901	-704.395255	-704.201579	-704.182943	-704.181869	-704.252196
ts102c1-103c1	-704.377437	-704.392517	-704.198834	-704.179613	-704.178539	-704.252001
ts100c2-102c2	-704.379370	-704.397637	-704.204189	-704.185383	-704.184309	-704.255672
ts102c2-107c2	-704.385107	-704.400000	-704.205811	-704.186627	-704.185554	-704.258438
ts100c1-104c1	-704.374639	-704.393635	-704.198651	-704.180436	-704.179362	-704.248922
ts104c1-105c1	-704.382330	-704.396952	-704.202643	-704.183479	-704.182405	-704.255643
ts110c2-104c2	-704.379389	-704.392837	-704.199639	-704.180463	-704.179389	-704.252548
ts104c2-105c2	-704.373517	-704.387395	-704.193712	-704.174491	-704.173417	-704.246879

β-Hydride elimination prior to carbocyclization.

TS name	$E_{el\ vac}$ (au)	ZPE $_{vac}$ (au)	E $_{vac}$ (au)	H $_{vac}$ (au)	G $_{vac}$ (au)
ts167c1-166c1	-802.452338	-802.255258	-802.233981	-802.232907	-802.309435

$E_{el\ vac}$, ZPE $_{vac}$, E $_{vac}$, H $_{vac}$, G $_{vac}$, $E_{el\ THF}$, ZPE $_{THF}$, E $_{THF}$, H $_{THF}$ and G $_{THF}$ have the same meanings as in Table 1.

APPENDIX A

The C++ Code Used in Chapter 2

```
/*
This programm reads the input file input.xyz of form

NumberOfAtoms
Epsilon Sigma
x1 y1 z1
x2 y2 z2
.
.
It performs Energy Minimizataion asuuming
Lennard-Jones potential between atoms.

The output is to standart stream.  It has form

x1 y1 z1
x2 y2 z2
.
.

This C++ code was tested using gcc version 2.96 20000731 (Red Hat Linux 7.1
2.96-81)
*/

#include <iostream.h>
#include <fstream.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <limits.h>

#define DIVISOR 2147483648.0

int ndims=3;
//int maxdims=3;


struct optins_list
{
 char comment[256];

 int natoms;
 int goodcoord;
 int sigma;
 int epsilon;

 int dimsfortravel;
 int dimsforquench;

 int way_to_optimize; //Steepest decent

 int way_to_travel;
 int travel_length_distribution; // 1*,  sqrt(N)*, or N*
 double travel_step_size;
 int travel_step_unit; // 0=Gaussian(Sqrt(N)*l), 1*,  sqrt(N)*, or N*
 int number_of_travel_steps; // const


 int way_to_quench;
 double quench_step_size;
 double quench_step_unit;// 1*,  sqrt(N)*, or N*
 double number_of_quench_steps;
 double quench_tolerance;
 double quench_tolerance_unit;// 1*,  sqrt(N)*, or N*

 double termination_energy;
 int reporiting_depth;
 int random_seed; //0 makes the system use time -1 same as 0+random start


};

optins_list options;





int isqrt(int x)
{
 int y;
 for (y=0; y*y<x; y++)
 {
 }
 return y;
}

double random_gaussian_number( double sigma)
{
 int i;
 double sum=-6.0;
 for (i=1; i<=12; i++)
 {
  sum += ((double)rand())/DIVISOR;
 }
 return sum*sigma;
}
```

```
/* Prints an MxN matrix a*/
void printmatrix(double a[], int M, int N)
{
 int i,j;
 for (i=0; i <= M -1; i++)
  {
   for (j=0; j <= N -1; j++)
   {
    cout<< a[N*i+j]<<"    ";

   }
   cout<<"\n";
  }
 cout<<"\n";

 return;
}

/*The sum of the squares of extra coordinates*/
double Wd(double coord[], int natoms, int goodcoord, int ncoord)
{
 int atom, dim;
 double sum=0.0;
 for (atom = 0; atom<natoms; atom++)
 {
  for (dim=goodcoord; dim<ncoord; dim++)
  {
   sum+=coord[atom*ncoord+dim]*coord[atom*ncoord+dim];
  }
 }

  return sum;
}

void finddellWd(double coord[], double dellWd[], int natoms, int goodcoord,
int ncoord)
{
 int atom, dim;

 for (atom = 0; atom<natoms; atom++)
 {
  for (dim=0; dim<goodcoord; dim++)
  {
   dellWd[atom*ncoord+dim]= 0;
  }

  for (dim=goodcoord; dim<ncoord; dim++)
  {
   dellWd[atom*ncoord+dim]= 2.0*coord[atom*ncoord+dim];
  }
 }

  return;
}
```

```
double power(double x, int y) //returns x^y, y is int >= 0
{ /*
 int i;

 double result=1;

 for ( i = 1;  i <= y;  i++)
 {
  result*=x;
 }

 return result;  */
 return pow(x,y);
}

//Returns distance between atoms at positiopns i1 and i2
double dist(double coord[],int i1, int i2)
{

 int dim;
 double dcoordinate;
 double sum=0;

 for (dim=0; dim<ndims; dim++)
 {
  dcoordinate=coord[i2*ndims+dim]-coord[i1*ndims+dim];
  sum += dcoordinate * dcoordinate;
 }
 return sqrt(sum);
}

//vector length
double vectorlength(double v[], int n)
{
 double sum=0.0;
 int k;//counter
 for ( k = 0;  k < n;  k++)
 {
  sum+=  v[k]*v[k];
 }
 return sqrt(sum);
}

//vector length
double vectorlongestcompont(double v[], int n)
{
 double longestcomponent=0.0;
 int k;//counter
 for ( k = 0;  k < n;  k++)
 {
  if (longestcomponent<fabs(v[k]))
  {
   longestcomponent =  fabs(v[k]);
  }
 }
 return longestcomponent;
}


double dotproduct(double v1[], double v2[], int n)
{
 double sum=0.0;
 int k;//counter
 for ( k = 0;  k < n;  k++)
 {
  sum+=  v1[k]*v2[k];
 }
 return sum;
}

//makes result = a*v
void vectormultipliedbyscalor(double result[], double a,
                                  double v[], int n)
{

 int k;//counter
 for ( k = 0;  k < n;  k++)
 {
  result[k] =  a*v[k];
 }
 return;
}


//makes result = v1+a*v2
void addvectormultipliedbyscalor(double result[], double v1[],double v2[],
                                  double a, int n)
{

 int k;//counter
 for ( k = 0;  k < n;  k++)
 {
  result[k] =  v1[k]+a*v2[k];
 }
 return;
}


//makes result = a*v1+b*v2
void addvectorsmultipliedbyscalors(double result[], double v1[],double v2[],
                                  double a, double b, int n)
{

 int k;//counter
 for ( k = 0;  k < n;  k++)
 {
  result[k] =  a*v1[k]+b*v2[k];
 }
 return;
}
```

```
/*Puts zeros in all n array elements */
void zeroarray(double a[], int n)
{
 int i;
 for (i=0;i<n;i++)
 {
  a[i]=0.0;
 }
}
//vector length
double furthestcoordinates(double coord[], int natoms, int ncoord)
{
 double longestcomponent=0.0;
 int atom, dim;
 double sum;

 for (atom = 0; atom<natoms; atom++)
 {
  sum =0.0;
  for (dim=0; dim<ncoord; dim++)
  {
   sum+=coord[atom*ncoord+dim]*coord[atom*ncoord+dim];
  }

  if (longestcomponent<sum)
  {longestcomponent =  sum; }

 }

 return sqrt(longestcomponent);
}


void go3toN(double coord3[], double coordN[], int natoms,
                                    int goodcoord, int ncoord)
{
int atom, dim;

 for (atom = 0; atom<natoms; atom++)
 {
  for (dim=0; dim<goodcoord; dim++)
  {
   coordN[atom*ncoord+dim]= coord3[atom*goodcoord+dim];
  }

  for (dim=goodcoord; dim<ncoord; dim++)
  {
   coordN[atom*ncoord+dim]=0.0;
  }
 }

 ndims=ncoord;
 return;
}


/*goes from an already uphysical system to an unphysical system of more
dimensions*/
void goNtobiggerN(double coord3[], double coordN[], int natoms,
                                    int ncoord, int newncoord)
{
int atom, dim;

 for (atom = 0; atom<natoms; atom++)
 {
  for (dim=0; dim<ncoord; dim++)
  {
   coordN[atom*newncoord+dim]= coord3[atom*ncoord+dim];
  }

  for (dim=ncoord; dim<newncoord; dim++)
  {
   coordN[atom*newncoord+dim]=0.0;
  }
 }

 ndims=newncoord;

 return;
}


void goNto3(double coordN[], double coord3[], int natoms,
                                    int goodcoord, int ncoord)
{
int atom, dim;

 for (atom = 0; atom<natoms; atom++)
 {
  for (dim=0; dim<goodcoord; dim++)
  {
   coord3[atom*goodcoord+dim]=coordN[atom*ncoord+dim];
  }
 }
 ndims=goodcoord;


 return;
}

void normalizevector(double v[], int n)
{
 int i;
 double length=vectorlength(v,n);

 for (i=0; i<n; i++)
 {
  v[i]/=length;
 }
```

```
}

/* Returnes true if the dot product of normalized vectors
is smaller than tolerance.

WARNONG! The way this was coded may not work on really big
and really small vectors because of representation errors
and overflows. It would be much safer (and slower) to
normalize the vectors first, and then to look for their dot product.
*/

bool vectors_are_perpendicular(double v1[], double v2[],
                               int n, double tolerance)
{

 return(fabs((
       dotproduct(v1, v2,n)
       /vectorlength(v1,n)
       /vectorlength(v2,n)
       ))<tolerance);

}


void partlysupressextracomponentsandnormalize(double coordN[], int natoms,
                                    int  goodcoord,  int  ncoord,  const  double
supressfactor)
{

 int atom, dim;

 for (atom = 0; atom<natoms; atom++)
 {
  for (dim=goodcoord; dim<ncoord; dim++)
  {
   coordN[atom*ncoord+dim]*=supressfactor;
  }
 }

 normalizevector(coordN,ncoord*natoms);

 return;
}


double findenergy(double coord[], int natoms,
                  double epsilon, double sigma)
{
 //double s6=power(sigma,6);//sigma^6
 //double s12=power(sigma,12);//sigma^12
 int k; //atom 1
 int l; //atom2

 double distance;
 double interaction;
 double result=0.0;


 for ( k = 0;  k < natoms-1;  k+=1)
 {
  for ( l = k+1;  l< natoms;   l+=1)
  {
   distance=dist(coord,k,l);
   interaction= 4*epsilon*(power(sigma/distance,12)-power(sigma/distance,6));
   result+= interaction;
  }
 }
 return  result;
}




void outputatoms(double coord[],int natoms,double epsilon, double sigma)
{
 int counter;//atom number
 int dim;
 for ( counter = 0;  counter < natoms;  counter++)
     {
      cout << "                  " ;
      for(dim=0; dim<ndims; dim++)
       {
         cout<<coord[ndims*counter+dim] << "  ";
       }
      cout <<endl;
     }

  //cout << "The energy of the interaction is "<<
   //  findenergy(coord, natoms, epsilon, sigma)<<endl;
}

void findgradient(double coord[], double dell[], int natoms,
                  double epsilon, double sigma)
{
 double s7=power(sigma,7);//sigma^7
 double s13=power(sigma,13);//sigma^13
 int k; //atom 1
 int l; //atom2
 int i; //direction
 double distance;
 double interaction;
 double dirinteraction;//interaction in one direction;

 //Fills all dell with zeros
 for ( k = 0;  k < natoms*ndims;  k++)
 {
     dell[k]=0;
 }

 for ( k = 0;  k < natoms-1;  k+=1)
 {
  for ( l = k+1;  l< natoms;   l+=1)
```

```
   {
    distance=dist(coord,k,l);
    interaction=                                                    4*epsilon*(-
12*s13/power(distance,13)+6*s7/power(distance,7));
  // cout<< distance<<endl;
    for  ( i = 0;  i < ndims;  i+=1)
    {
     dirinteraction= interaction*(coord[ndims*l+i]-coord[ndims*k+i])/distance;
     dell[ndims*k+i]+=dirinteraction;
     dell[ndims*l+i]-=dirinteraction;
    }
   }
  }
}

void find_gradientE(double coord[], double dell[], int natoms,
                  double epsilon, double sigma)
{
 int k;
 findgradient( coord,  dell,  natoms,
                   epsilon,  sigma);
 for ( k = 0;  k < natoms*ndims;  k++)
 {
     dell[k]*=-1;
 }
 return;
}

/*copies a to b, so
b=a*/
void copyarray(double a[], double b[], int n)
{
 int i;
 for (i=0;i<n;i++)
 {
   b[i]=a[i];
 }
 return;
}

/*Changes coord to a random coordinate on the face
   of a dims dimentional sphere*/
void randompointonunitsphere(double coord[],
                             int dims/*number of coordinates*/)
{
 int i;
 do
 {
  for (i=0; i<dims; i++)
  {
   coord[i]=((double)rand())/DIVISOR*2.0-1.0;
  }
 } while(0 /*vectorlength(coord,dims)>1.0*/);

 normalizevector(coord,dims);
 return;
}
```

```
/*
Moves one particle in space.
*/
void randomparticlemove(double coord[], int natoms,
                        int ncoord,double sigma)
{
 int i;
 int atom=rand()%natoms;//atom to be moved
 for (i=atom*ncoord; i<(atom+1)*ncoord; i++)
  {
   coord[i]+=0.1*sigma*(((double)rand())/DIVISOR*2.0-1.0);
  }

 return;
}

/*Corrects
 */
void correctpositiontochangeenergy(double coord[], int natoms,
                                    int goodcoord, int ncoord,
                                    double epsilon, double sigma,
                                    double E0,/*current energy*/
                                    const double E1/*the energy we want*/)
{
 const int maxpossibleiterations=1000*natoms;
 /*Because this search is newton-like, it has a possibility
    of not converging and freezing up the programm for that reason
    after a ceratin number of iterations it will just give up.
    */
 double dellE[natoms*ncoord];  //  gradiant of enegrgy
 double energyslope;
 double displacement[natoms*ncoord];

 int iterations=0; //number of elapsed iterations

 do
  {
   findgradient(coord, dellE, natoms, epsilon, sigma);
   energyslope = vectorlength(dellE,natoms*ncoord);

   copyarray(dellE,displacement,natoms*ncoord);
   normalizevector(displacement,natoms*ncoord);
   addvectormultipliedbyscalor(coord, coord,displacement,
                             (E0-E1)/energyslope, natoms*ncoord);

   E0=findenergy(coord, natoms, epsilon,  sigma);

   iterations++;
  }while( (fabs(E0-E1)>0.001)&&(iterations<=maxpossibleiterations));
  if (iterations>maxpossibleiterations)
  {cout<<"Giving up in  correctpositiontochangeenergy after "<<
            iterations<<
            "  iterations."<<endl;}
  //cout<<"In correctpositiontochangeenergy iterations = "<<iterations<<endl;
 return;
}
/* Lowers the energy and tries to go to a lower min*/
void lowerenergywithextradims(double coord[], int natoms,
                                    int goodcoord, int ncoord,
                                    double epsilon, double sigma,
                                    double originalenergy)
{


  double Enew= originalenergy -0.1 ;
  correctpositiontochangeenergy(coord,natoms,
                                     goodcoord,  ncoord,
                                     epsilon,  sigma, originalenergy, Enew);

}
/* Makes a random step with all atoms, but so that the system
remains at constant energy*/
void walkinsomedirectionatconstantenergy(double coord[], int natoms,
                                    int goodcoord, int ncoord,
                                    double epsilon, double sigma,
                                    double originalenergy)
{




 int maxsteps;
 /*=50*natoms;*/
 //travel_step_unit;// 1*,  sqrt(N)*, or N*
 switch ( options.travel_length_distribution)
 {
  case 1:
   maxsteps=options.number_of_travel_steps;
   break;
  case 2:
   maxsteps=isqrt(natoms)*options.number_of_travel_steps;
   break;
  case 3:
   maxsteps=natoms*options.number_of_travel_steps;
   break;
 }
 //int travel_length_distribution; // const, Gaussian

 double steplength  /*0.1*sigma*/;

 switch (options.travel_step_unit)
 {
  case 1:
   steplength=options.travel_step_size;
   break;
  case 2:
   steplength=options.travel_step_size*sqrt(natoms);
   break;
  case 3:
   steplength=options.travel_step_size*natoms;
   break;







  case 0:
   steplength = random_gaussian_number(options.travel_step_size);
   break;
 }



 double E1;

 int atom;
 int dim;

 double dellWd[natoms*ncoord]; // here  dellWd is a direction
 randompointonunitsphere(dellWd,natoms*ncoord/*number of coordinates*/);
 partlysupressextracomponentsandnormalize(dellWd,  natoms,
                                     goodcoord,  ncoord,  1.0);


 double dellE[natoms*ncoord];  // normalized gradiant of enegrgy
 int i;
 double prevwidth=0.0;//width in the previous step;
 for (i=0; i<maxsteps; i++)
 {
  prevwidth = Wd(coord,natoms, goodcoord, ncoord);
  findgradient(coord, dellE, natoms, epsilon, sigma);
  normalizevector(dellE, natoms*ncoord);

  // component of gradiant of width parallel to gradiant of energy
  double We[natoms*ncoord];
  double dotproductofdellEandnormalizeddellWd =
  dotproduct(dellWd,dellE,natoms*ncoord);

  vectormultipliedbyscalor(We, dotproductofdellEandnormalizeddellWd,
                                   dellE, natoms*ncoord);

  double dWd[natoms*ncoord];
  //makes result = v1+a*v2
  addvectormultipliedbyscalor(dWd, dellWd,We,-1.0, natoms*ncoord);

  addvectormultipliedbyscalor(coord, coord,dWd, steplength, natoms*ncoord);


  E1=findenergy(coord, natoms, epsilon,  sigma);

  //outputatoms(coord, natoms, epsilon,  sigma);
  correctpositiontochangeenergy(coord,natoms,
                                     goodcoord,  ncoord,
                                     epsilon,  sigma, E1, originalenergy);
  //The following two lines adjust the direction of travel to prevent travel
into atoms.
  copyarray(dWd, dellWd,natoms*ncoord); //dellWd=dWd;

  //normalizevector(dellWd, natoms*ncoord);

  //When width starts decreacing again, the destination was reached.
  /*
  if (Wd(coord,natoms, goodcoord, ncoord)<prevwidth)
   {i+=maxsteps/10;
    cout<<                                                                   "
got over the hill"<< endl;
   }*/

 }
 //outputatoms(coord, natoms, epsilon,  sigma);
 return;
}

/* Makes a random step with all atoms, but so that the system
remains at constant energy*/
void walk_in_somedirection_at_constant_energy_for_fixed_number_of_steps(
                                    double coord[], int natoms,
                                    int goodcoord, int ncoord,
                                    double epsilon, double sigma,
                                    double originalenergy)
{

 const double steplength = 0.1*sigma /*0.1*sigma*/;
 const int maxsteps=50*natoms;

 double E1;

 int atom;
 int dim;

 double dellWd[natoms*ncoord]; // here  dellWd is a direction
 randompointonunitsphere(dellWd,natoms*ncoord/*number of coordinates*/);
 partlysupressextracomponentsandnormalize(dellWd,  natoms,
                                     goodcoord,  ncoord,  0.1);


 double dellE[natoms*ncoord];  // normalized gradiant of enegrgy
 int i;
 double prevwidth=0.0;//width in the previous step;
 for (i=0; i<maxsteps; i++)
 {
  prevwidth = Wd(coord,natoms, goodcoord, ncoord);
  findgradient(coord, dellE, natoms, epsilon, sigma);
  normalizevector(dellE, natoms*ncoord);

  // component of gradiant of width parallel to gradiant of energy
  double We[natoms*ncoord];
  double dotproductofdellEandnormalizeddellWd =
  dotproduct(dellWd,dellE,natoms*ncoord);
  vectormultipliedbyscalor(We, dotproductofdellEandnormalizeddellWd,
                                   dellE, natoms*ncoord);

  double dWd[natoms*ncoord];
  //makes result = v1+a*v2
  addvectormultipliedbyscalor(dWd, dellWd,We,-1.0, natoms*ncoord);

  addvectormultipliedbyscalor(coord, coord,dWd, steplength, natoms*ncoord);


  E1=findenergy(coord, natoms, epsilon,  sigma);

  //outputatoms(coord, natoms, epsilon,  sigma);
  correctpositiontochangeenergy(coord,natoms,
                                     goodcoord,  ncoord,
                                     epsilon,  sigma, E1, originalenergy);

  //The following two lines adjust the direction of travel to prevent travel
into atoms.
  copyarray(dWd,dellWd,natoms*ncoord);//dellWd=dWd;

  //normalizevector(dellWd, natoms*ncoord);

  //When width starts decreacing again, the destination was reached.
  /*
  if (Wd(coord,natoms, goodcoord, ncoord)<prevwidth)
   {i+=maxsteps/10;
    cout<<                                                                  "
got over the hill"<< endl;
   }
  */
 }
 //outputatoms(coord, natoms, epsilon,  sigma);
 return;
}
/* Makes a random step with all atoms, but so that the system
remains at constant energy*/
void testwalk(double coord[], int natoms,
                                    int goodcoord, int ncoord,
                                    double epsilon, double sigma,
                                    double originalenergy)
{

 const double steplength = 0.1*sigma /*0.1*sigma*/;
 const int maxsteps=500;

 double E1;

 int atom;
 int dim;

 double dellWd[natoms*ncoord]; // here  dellWd is a direction
 randompointonunitsphere(dellWd,natoms*ncoord/*number of coordinates*/);



 double dellE[natoms*ncoord];  // normalized gradiant of enegrgy
 int i;
 double prevwidth=0.0;//width in the previous step;
 for (i=0; i<maxsteps; i++)
 {
  prevwidth = Wd(coord,natoms, goodcoord, ncoord);
  findgradient(coord, dellE, natoms, epsilon, sigma);
  normalizevector(dellE, natoms*ncoord);

  // component of gradiant of width parallel to gradiant of energy
  double We[natoms*ncoord];
  double dotproductofdellEandnormalizeddellWd =
  dotproduct(dellWd,dellE,natoms*ncoord);

  vectormultipliedbyscalor(We, dotproductofdellEandnormalizeddellWd,
                                   dellE, natoms*ncoord);

  double dWd[natoms*ncoord];
  //makes result = v1+a*v2
  addvectormultipliedbyscalor(dWd, dellWd,We,-1.0, natoms*ncoord);

  addvectormultipliedbyscalor(coord, coord,dWd, steplength, natoms*ncoord);


  E1=findenergy(coord, natoms, epsilon,  sigma);

  //outputatoms(coord, natoms, epsilon,  sigma);
  correctpositiontochangeenergy(coord,natoms,
                                   goodcoord,  ncoord,
                                   epsilon,  sigma, E1, originalenergy);

  //The following two lines adjust the direction of travel to prevent travel
into atoms.
  copyarray(dWd,dellWd,natoms*ncoord);//dellWd=dWd;

  //normalizevector(dellWd, natoms*ncoord);

  //When width starts decreacing again, the destination was reached.
  cout << "              i = "<<i;
  cout << "     width = "<<Wd(coord,natoms, goodcoord, ncoord)<<endl;
  if (/*i==(maxsteps/4) &&*/ (Wd(coord,natoms, goodcoord, ncoord)>prevwidth))
   {
    partlysupressextracomponentsandnormalize(dellWd,  natoms,
                                   goodcoord,  ncoord,  -1.0);

   }

 }
 //outputatoms(coord, natoms, epsilon,  sigma);
 return;
}


/* Makes a random step with all atoms, but so that the system
remains at constant energy*/
void makerandomstepatconstantenergy(double coord[], int natoms,
                                    int goodcoord, int ncoord,
                                    double epsilon, double sigma,
                                    double originalenergy)
{
 const double steplength = natoms*0.1*sigma /*0.1*sigma*/;


 double E1;

 int atom;
 int dim;

 double dellE[natoms*ncoord];  // normalized gradiant of enegrgy
 findgradient(coord, dellE, natoms, epsilon, sigma);
 normalizevector(dellE, natoms*ncoord);

 double dellWd[natoms*ncoord];
 randompointonunitsphere(dellWd,natoms*ncoord/*number of coordinates*/);


 /*
 double normalizeddellWd[natoms*ncoord];  normalizedgradiant of width
 copyarray(dellWd,normalizeddellWd,natoms*ncoord);
 normalizevector(normalizeddellWd, natoms*ncoord);
 */
 // component of gradiant of width parallel to gradiant of energy
 double We[natoms*ncoord];
 double dotproductofdellEandnormalizeddellWd =
  dotproduct(dellWd,dellE,natoms*ncoord);

 vectormultipliedbyscalor(We, dotproductofdellEandnormalizeddellWd,
                                  dellE, natoms*ncoord);

 double dWd[natoms*ncoord];
 //makes result = v1+a*v2
 addvectormultipliedbyscalor(dWd, dellWd,We,-1.0, natoms*ncoord);

 addvectormultipliedbyscalor(coord, coord,dWd, steplength, natoms*ncoord);

 //cout                        <<                         dotproduct(dWd,dellE,
natoms*ncoord)/vectorlength(dWd,natoms*ncoord)<<endl;

 E1=findenergy(coord, natoms, epsilon,  sigma);
 correctpositiontochangeenergy(coord,natoms,
                                    goodcoord,  ncoord,
                                    epsilon,  sigma, E1, originalenergy);

 return;
}


/*moves oneparticle at constantenergy
(Well other particles also move a little.)*/
void make1particlestepatconstantenergy(double coord[], int natoms,
                                    int goodcoord, int ncoord,
                                    double epsilon, double sigma,
                                    double originalenergy)
{
 randomparticlemove( coord, natoms,ncoord, sigma);
 //energy after the move
 double E1=findenergy(coord, natoms, epsilon,  sigma);
 correctpositiontochangeenergy(coord,natoms,
                                   goodcoord,  ncoord,
                                   epsilon,  sigma, E1, originalenergy);
 return;
}

void travelND(double coord[], int natoms,
              int goodcoord, int ncoord,
              double epsilon, double sigma)
{

 double originalenergy=findenergy(coord, natoms, epsilon,  sigma);
 int i;
 for (i=0; i<1/*100*/; i++)
 {
  /*
  make1particlestepatconstantenergy( coord,  natoms,
                                   goodcoord, ncoord,
                                   epsilon,  sigma, originalenergy);

  makerandomstepatconstantenergy( coord,  natoms,
                                   goodcoord, ncoord,
                                   epsilon,  sigma, originalenergy);
                                   */
  walkinsomedirectionatconstantenergy( coord,  natoms,
                                   goodcoord, ncoord,
                                   epsilon,  sigma, originalenergy);

  /*
  walk_in_somedirection_at_constant_energy_for_fixed_number_of_steps(
                                   coord,  natoms,
                                   goodcoord, ncoord,
                                   epsilon,  sigma, originalenergy);
  testwalk( coord,  natoms,
                                   goodcoord, ncoord,
                                   epsilon,  sigma, originalenergy);

  lowerenergywithextradims( coord,  natoms,
                                   goodcoord, ncoord,
                                   epsilon,  sigma, originalenergy); */


 }


 return;
}



/*returnes the component of the gradiant of the width function
perpendicular to the gradiant of the energy function     */
void dellWdatconstantenergy(double coord[], int natoms,
                                   double dWd[],
                                   int goodcoord, int ncoord,
                                    double epsilon, double sigma)
{
 int atom;
 int dim;

 double dellE[natoms*ncoord];  // normalized gradiant of enegrgy
 findgradient(coord, dellE, natoms, epsilon, sigma);
 normalizevector(dellE, natoms*ncoord);

 double dellWd[natoms*ncoord];  //gradiant of width
 finddellWd(coord,  dellWd,  natoms,  goodcoord,  ncoord);
 /*
 double normalizeddellWd[natoms*ncoord];  normalizedgradiant of width
 copyarray(dellWd,normalizeddellWd,natoms*ncoord);
 normalizevector(normalizeddellWd, natoms*ncoord);
 */
 // component of gradiant of width parallel to gradiant of energy
 double We[natoms*ncoord];
 double dotproductofdellEandnormalizeddellWd =
  dotproduct(dellWd,dellE,natoms*ncoord);

 vectormultipliedbyscalor(We, dotproductofdellEandnormalizeddellWd,
                                 dellE, natoms*ncoord);


 //makes result = v1+a*v2
 addvectormultipliedbyscalor(dWd, dellWd,We,-1.0, natoms*ncoord);
 normalizevector(We, natoms*ncoord);
 /*cout<<"dotproduct "<<dotproduct(We,dellE,natoms*ncoord)<<endl;*/
 return;
}

/*(E-Eo)^2+W*/
double compressionfunction(double coord[], int natoms,
                        int goodcoord, int ncoord,
                        double epsilon, double sigma,
                        double E0)
{
 double energypart =
  findenergy(coord, natoms, epsilon,  sigma) - E0;
 return ((energypart*energypart)+
            Wd( coord, natoms, goodcoord, ncoord));
}


void finddellcompressionfunctionNUM(double coord[], int natoms,
                        int goodcoord, int ncoord,
                        double epsilon, double sigma,
                        double E0, double dellF[])
{
 const double dx=0.0001;
 double coord2[natoms*ncoord];
 copyarray(coord,coord2,natoms*ncoord);
 int i;
 for (i=0; i<natoms*ncoord;i++)
 {
  coord2[i]+=dx;
  dellF[i]=(compressionfunction( coord2,  natoms,
                       goodcoord,  ncoord,
                       epsilon,  sigma,
                       E0)-
          compressionfunction( coord,  natoms,
                       goodcoord,  ncoord,
                       epsilon,  sigma,
                       E0))/dx;
  coord2[i]=coord[i];
 }

}
void finddellenergypartNUM(double coord[], int natoms, int ncoord,
               double E0,
               double epsilon, double sigma, double dellEP[])
{
 const double dx=0.0001;
 double coord2[natoms*ncoord];
 copyarray(coord,coord2,natoms*ncoord);

 double EP1=(findenergy(coord, natoms, epsilon,  sigma) - E0)
          *(findenergy(coord, natoms, epsilon,  sigma) - E0);
 double EP2;

 int i;
 for (i=0; i<natoms*ncoord;i++)
 {
  coord2[i]+=dx;
  EP2=(findenergy(coord2, natoms, epsilon,  sigma) - E0)
     *(findenergy(coord2, natoms, epsilon,  sigma) - E0);

  dellEP[i]=(EP2-EP1)/dx;
  coord2[i]=coord[i];
 }

}

void finddellWNUM(double coord[], int natoms, int ncoord,
               int goodcoord,
               double epsilon, double sigma, double dellW[])
{
 const double dx=0.0001;
 double coord2[natoms*ncoord];
 copyarray(coord,coord2,natoms*ncoord);

 double W1=Wd( coord, natoms, goodcoord, ncoord);
 double W2;

 int i;
 for (i=0; i<natoms*ncoord;i++)
 {
  coord2[i]+=dx;
  W2=Wd( coord2, natoms, goodcoord, ncoord);

  dellW[i]=(W2-W1)/dx;
  coord2[i]=coord[i];
 }
```

```
}

void findgradientNUM(double coord[],double dellE[],
        int ncoord, int natoms, double epsilon, double sigma)
{

 const double dx=0.0001;
 double coord2[natoms*ncoord];
 copyarray(coord,coord2,natoms*ncoord);

 double Y1=findenergy(coord, natoms, epsilon,  sigma);
 double Y2;

 int i;
 for (i=0; i<natoms*ncoord;i++)
 {
  coord2[i]+=dx;
  Y2=findenergy(coord2, natoms, epsilon,  sigma);

  dellE[i]=(Y2-Y1)/dx;
  coord2[i]=coord[i];
 }


}

/*Gradient of the compression function*/
void finddellcompressionfunction(double coord[], int natoms,
                        int goodcoord, int ncoord,
                        double epsilon, double sigma,
                        double E0, double dellF[])
{
  double energypart =
  findenergy(coord, natoms, epsilon,  sigma) - E0;

  double dellE[natoms*ncoord];
  findgradient(coord, dellE, natoms, epsilon, sigma);

  double dellWd[natoms*ncoord];  //gradiant of width
  finddellWd(coord,  dellWd,  natoms,  goodcoord,  ncoord);

 /*makes result = v1+a*v2
void addvectormultipliedbyscalor(double result[], double v1[],double v2[],
                                  double a, int n) */

  addvectormultipliedbyscalor(dellF,dellWd,dellE,
                           -2.0*energypart,natoms*ncoord);
 /*
 int i;
 for (i=0; i<natoms*ncoord;i++)
 {
  dellF[i]=-dellE[i]*2.0*energypart+dellWd[i];
 }
*/

}
```

```
double  compressionFatlengthindirection(double  coord[],  double  direction[],
int natoms,
                 double  epsilon,  double  sigma,  int  goodcoord,  int  ncoord,
double Eo,
                 double l)
{

 double newcoord[natoms*ndims];
 /*Extra array elements are made zero.
 zeroextracoordinates(newcoord, natoms);
 zeroextracoordinates(direction, natoms);*/
 addvectormultipliedbyscalor(newcoord, coord,direction,
                                 l, natoms*ndims);
 return compressionfunction( newcoord,  natoms,
                         goodcoord,  ncoord,
                         epsilon,  sigma,
                         Eo);
}


void inversematrixelements(double matrix[], int n)
{
 int i;
 for(i=0; i<n; i++)
 {
  matrix[i]=1.0/matrix[i];
 }
}

void   increasesmallelements(double   matrix[],   double   bigvalue,   double
smallvalue, int n)
{
 int i;
 for  (i=0; i<n; i++)
 {
   if  (fabs(matrix[i]<smallvalue))
    {matrix[i]=bigvalue;}
 }
 return;
}



double optimizeFonline(double coord[], double direction[],int natoms,
                        int goodcoord, int ncoord,
                        double epsilon, double sigma,double E0,
                        double originaldistance, int maxcycles)
{
    double movedthisfar;
    double leftmultiplier=0.0;
    double rightmultiplier=originaldistance;
    double midmultiplier= 0.5*(leftmultiplier+rightmultiplier);

    int ncoords= ncoord * natoms; //array size
    double dell[ncoords];
    finddellcompressionfunction( coord, natoms,
                        goodcoord,  ncoord,
                        epsilon,  sigma,
                        E0,  dell);
    //finddellWd( coord, dell,  natoms,  goodcoord,  ncoord);

    //find_gradientE(coord, dell, natoms, epsilon, sigma);
    if (dotproduct(direction, dell, ncoords)>0.0)
    {
     vectormultipliedbyscalor(direction, -1.0,
                              direction, ncoords);
    }

    int i;
    for (i=0; i<maxcycles; i++)
    {
     midmultiplier= 0.5*(leftmultiplier+rightmultiplier);
     if (compressionFatlengthindirection(coord, direction ,  natoms,
                     epsilon,  sigma,  goodcoord,  ncoord,  E0,
                     leftmultiplier)
                     >
           compressionFatlengthindirection(coord, direction,  natoms,
                     epsilon,  sigma,  goodcoord,  ncoord,  E0,
                     rightmultiplier)
                      )
     {
       leftmultiplier=midmultiplier;
       movedthisfar=midmultiplier; //cout<<"moving right"<<endl;
     }
     else
     {
       rightmultiplier=midmultiplier;
       movedthisfar=midmultiplier;
       //cout<<"moving left"<<endl;
     }
    }

  return movedthisfar;
}

double biggest_extra_coordinate(double coord[], int natoms,
                                   int goodcoord, int ncoord)
{
 double result=0.0;
 int atom, extracoordinate, currentcoordinate;
 for (atom=0; atom<natoms; atom++)
 {
  for (extracoordinate=goodcoord; extracoordinate<ncoord; extracoordinate++)
  {
   currentcoordinate=(atom*ncoord)+extracoordinate;
   if (fabs(coord[currentcoordinate])>result)
   {
     result=fabs(coord[currentcoordinate]);
   }
  }
 }
 return result;
```

```
}


int cgoptWDatConstantE(double coord[], int natoms,
                double epsilon, double sigma,
                int goodcoord, int ncoords/*number of dimensions*/,
                double steplength,
                double maxgradiant /*When gradient
                smaller than this is reached, the
                search terminates*/)

{

 const int ITMAX=20000;

 int ncoord=natoms*ncoords;
 int j, its, iter;
 double gg, gam, fp, dgg, fret;
 double g[ncoord], h[ncoord], x[ncoord], xi[ncoord];
 int n=ncoord;
 double distancetosample=natoms*0.1;
 double movedthisfar;
 const double E0=findenergy(coord, natoms, epsilon,  sigma);
 //fp=findenergy(coord,natoms, epsilon,  sigma);

 fp=compressionfunction( coord,  natoms,
                         goodcoord,  ncoords,
                         epsilon,  sigma,
                         E0);
 //find_gradientE(coord, xi, natoms, epsilon, sigma);

 finddellcompressionfunction( coord, natoms,
                        goodcoord,  ncoords,
                        epsilon,  sigma,
                        E0,  xi);
 for (j=0;j<ncoord;j++)
 {
  g[j]= -xi[j];
  xi[j] = h[j] = g[j];
 }


 for (its=0; its<ITMAX; its++)
 {

  iter=its;
  {


   movedthisfar=optimizeFonline(coord,  xi, natoms,
                         goodcoord, ncoords,
                         epsilon,  sigma, E0,
                        distancetosample, 10);

/*   movedthisfar=optimizeEonline2(coord, xi, natoms,
                          ncoord,
                         epsilon,  sigma,
                        distancetosample, 10);  */

   distancetosample=movedthisfar*10.0;
   //cout<<"movedthisfar = "<<movedthisfar<<"   "<<flush;

   addvectormultipliedbyscalor(coord, coord, xi, movedthisfar, ncoord);
   for (j=0;j<ncoord;j++)
   {
    xi[j]*=movedthisfar;
   }
   fret=findenergy(coord,natoms, epsilon,  sigma);

  }

  fp=fret;

  finddellcompressionfunction( coord, natoms,
                        goodcoord,  ncoords,
                        epsilon,  sigma,
                        E0,  xi);
  //find_gradientE(coord, xi, natoms, epsilon, sigma);
  /*cout<<"gradient "<<vectorlength(xi,ncoord)<<
    ",   E "<<findenergy(coord,natoms, epsilon,  sigma)<<
    ",   l "<<distancetosample<<endl<<flush ; */
 //cout<<"biggest_extra_coordinate     "<<     biggest_extra_coordinate(coord,
natoms,
 //    goodcoord, ncoords)<<endl<<flush;

  if ((biggest_extra_coordinate(coord,  natoms,
     goodcoord, ncoords)<sigma*0.05)
     || (vectorlongestcompont(xi,ncoord)<maxgradiant))
   {
    //printmatrix(coord,natoms,ncoords);
    if (biggest_extra_coordinate(coord,  natoms,
     goodcoord, ncoords)<sigma*0.3)
    {return 1;}
    else
    {return 0;}
   }

  dgg=gg=0.0;
  for (j=0;j<ncoord;j++)
   {
    gg+=g[j]*g[j];
    //dgg+=xi[j]*xi[j]; //Fletcher-Reeves
    dgg+=(xi[j]+g[j])*xi[j]; //Polak-Ribiere
   }

  if (gg==0.0)
   return 1;
  gam=dgg/gg;
  for (j=0;j<n;j++)
  {
   g[j]= -xi[j];
   xi[j]=h[j]=g[j]+gam*h[j];
  }
 }
 return 1;
}




int  compressto3DbyTSwalk(double coord[], int natoms,
                        int goodcoord, int ncoord,
                        double epsilon, double sigma)
{
 cout << "                                                           E before
compressto3DbyTSwalk = "
      << findenergy(coord, natoms, epsilon,  sigma) << endl;
 const double E0=findenergy(coord, natoms, epsilon,  sigma);
 const int maxpossibleiterations=1000*natoms;
 double dellF[natoms*ncoord];  //  gradiant of the compression function
 double dellW[natoms*ncoord];  //  gradiant of the width function
 double direction[natoms*ncoord];

 double Fslope, currentF, currentW;
 int iterations=0; //number of elapsed iterations
 do
  {

   finddellcompressionfunction( coord, natoms,
                        goodcoord,  ncoord,
                        epsilon,  sigma,
                        E0,  dellF);
   Fslope = vectorlength(dellF,natoms*ncoord);
   normalizevector(dellF,natoms*ncoord);

   finddellWd(coord, dellW, natoms, goodcoord, ncoord);
   normalizevector(dellW,natoms*ncoord);

   //makes result = v1+a*v2
   addvectormultipliedbyscalor(direction,dellF,dellW,
                                  +0.999,  natoms*ncoord);

   normalizevector(direction,natoms*ncoord);

   addvectormultipliedbyscalor(coord,coord,direction,
                                     -0.01,natoms*ncoord);

   currentF= compressionfunction( coord,  natoms,
                         goodcoord,  ncoord,
                         epsilon,  sigma,
                         E0);
   currentW=Wd( coord, natoms, goodcoord, ncoord);
   /*
   cout<<"F = "<<currentF
    <<"   W = "<<currentW
    <<"   E = "<<findenergy(coord, natoms, epsilon,  sigma) - E0
    <<"   Fslope = "<< Fslope
    <<endl;*/
   iterations++;
  }while( (fabs(currentF)>0.0005*natoms)&&(iterations<=maxpossibleiterations)
                &&(currentW>0.0005*natoms));


 return 1;
}


int  compressto3DbyfastTSwalk(double coord[], int natoms,
                        int goodcoord, int ncoord,
                        double epsilon, double sigma)
{
 cout << "                                                              E before
compressto3DbyfastTSwalk = "
      << findenergy(coord, natoms, epsilon,  sigma) << endl;
 const double E0=findenergy(coord, natoms, epsilon,  sigma);
 const int maxpossibleiterations=1000*natoms;
 double dellF[natoms*ncoord];  //  gradiant of the compression function
 double dellW[natoms*ncoord];  //  gradiant of the width function
 double direction[natoms*ncoord];

 double Fslope, currentF, currentW, prevF, prevW;
 double maxcomponentmove=0.1*sigma;

 const double sucessmultiplier = 1.2;
 const double faluremultiplier = 0.5;

 int iterations=0; //number of elapsed iterations

 currentF= compressionfunction( coord,  natoms,
                         goodcoord,  ncoord,
                         epsilon,  sigma,
                         E0);
 currentW=Wd( coord, natoms, goodcoord, ncoord);

 do
  {

   finddellcompressionfunction( coord, natoms,
                        goodcoord,  ncoord,
                        epsilon,  sigma,
                        E0,  dellF);
   Fslope = vectorlength(dellF,natoms*ncoord);
   normalizevector(dellF,natoms*ncoord);

   finddellWd(coord, dellW, natoms, goodcoord, ncoord);
   normalizevector(dellW,natoms*ncoord);

   //makes result = v1+a*v2
   addvectormultipliedbyscalor(direction,dellW,dellF,
                                  +0.999,  natoms*ncoord);

   normalizevector(direction,natoms*ncoord);

   addvectormultipliedbyscalor(coord,coord,direction,
     -maxcomponentmove*vectorlongestcompont(direction,natoms*ncoord),
     natoms*ncoord);

   prevF=currentF;
   prevW=currentW;
   currentF= compressionfunction( coord,  natoms,
                         goodcoord,  ncoord,
                         epsilon,  sigma,
                         E0);
   currentW=Wd( coord, natoms, goodcoord, ncoord);

   if (dotproduct(dellF,dellW,natoms*ncoord)> -0.9)
   {
    if (currentF<prevF)
    {
     maxcomponentmove*=sucessmultiplier;
    }

    else
    {
     maxcomponentmove*=faluremultiplier;
    }
   }
   else
   {
    maxcomponentmove*=sucessmultiplier;
   }



   if (maxcomponentmove> biggest_extra_coordinate( coord,  natoms,
              goodcoord,  ncoord))
   {
    maxcomponentmove=biggest_extra_coordinate( coord,  natoms,
              goodcoord,  ncoord);
   }

   /*
   cout<<"F = "<<currentF
    <<"   W = "<<currentW
    <<"   E = "<<findenergy(coord, natoms, epsilon,  sigma) - E0
    <<"   Fslope = "<< Fslope
    <<endl;*/

  cout<<"               maxcomponentmove = "<< maxcomponentmove<<endl;

   iterations++;
  }while( biggest_extra_coordinate( coord,  natoms,
              goodcoord,  ncoord)>maxcomponentmove);

  cout<<"               iterations = "<< iterations <<endl;
  cout<<"               maxcomponentmove = "<< maxcomponentmove<<endl;

 return 1;
}
```

```
/*This procedure is dimensional compression at constant energy,
done as finding the root of a function (E-Eo)^2+W*/
int  compressto3DbyCG(double coord[], int natoms,
                        int goodcoord, int ncoord,
                        double epsilon, double sigma)
{
 cout << "                                                        E before
compressto3DbyCG = "
      << findenergy(coord, natoms, epsilon,  sigma) << endl;
 const double E0=findenergy(coord, natoms, epsilon,  sigma);

 /* Prints an MxN matrix a*/
 // printmatrix(coord, natoms, ncoord);


 const int maxpossibleiterations=1000*natoms;
 /*Because this search  has a possibility
    of not converging and freezing up the programm for that reason
    after a ceratin number of iterations it will just give up.
    */
 double dellF[natoms*ncoord];  //  gradiant
 double  prevdellFsquared=1e30;  //     suqre  of  length  of  previous  energy
gradient

 /*Vars about the function being optimized*/
 double Fslope, currentF, prevF;
 double displacement[natoms*ncoord];
 double d[natoms*ncoord];
 double prevd[natoms*ncoord];
 zeroarray(prevd,natoms*ncoord);

 /*This variable is for constraining the distance
  Newton's algorithm can jump. It is there, so it
  would not diverge*/
 double maxdistancetomove = 0.1;
 double maxmaxdistancetomove = 0.3*natoms;
 int iterations=0; //number of elapsed iterations

 currentF= compressionfunction( coord,  natoms,
                         goodcoord,  ncoord,
                         epsilon,  sigma,
                         E0);

 double distancetomove=0.1;
 double beta=0.0;
 bool firstiteration=true; //true only in the first iteration.
 do
  {

   finddellcompressionfunction( coord, natoms,
                        goodcoord,  ncoord,
                        epsilon,  sigma,
                        E0,  dellF);



   Fslope = vectorlength(dellF,natoms*ncoord);
   beta=Fslope*Fslope/prevdellFsquared;
  /*makes result = v1+a*v2
void addvectormultipliedbyscalor(double result[], double v1[],double v2[],
                                  double a, int n) */


   //addvectormultipliedbyscalor(d, double v1[],double v2[],
   //                              double a,  natoms*ncoord)
   if (!firstiteration)
   {
    int i;
    int nelements=natoms*ncoord;
    for (i=0; i<nelements;i++)
    {
      d[i]=-dellF[i] + (beta*prevd[i]);
    }
   }
   else
   {
    int i;
    int nelements=natoms*ncoord;
    for (i=0; i<nelements;i++)
    {
      d[i]=-dellF[i];
    }
   }

   copyarray(d,displacement,natoms*ncoord);
   normalizevector(displacement,natoms*ncoord);

   if(distancetomove>maxdistancetomove)
   {distancetomove=maxdistancetomove;}

   double multiplier=3.0;

   do
   {
    multiplier*=0.5;
    /*copies a to b, so b=a*/
    //void copyarray(double a[], double b[], int n)

   }  while(currentF<=compressionFatlengthindirection(coord,  displacement  ,
natoms,
                 epsilon,  sigma,  goodcoord,  ncoord,  E0,
                 multiplier*distancetomove));
   //  cout<<"multiplier*distancetomove  =  "<<    multiplier*distancetomove
<<endl;

    double movedthisfar=optimizeFonline(coord, displacement, natoms,
                         goodcoord,  ncoord,
                         epsilon,  sigma, E0,
                         multiplier*distancetomove, 15);

     //cout<<"movedthisfar = "<<   movedthisfar<<endl;
   /*{
```

```
 double leftmultiplier=0.0;
 double rightmultiplier=multiplier*2.0;
 double midmultiplier= 0.5*(leftmultiplier+rightmultiplier);

 int i;
 for (i=0; i<15; i++)
 {
  midmultiplier= 0.5*(leftmultiplier+rightmultiplier);
  if (compressionFatlengthindirection(coord, displacement ,  natoms,
                 epsilon,  sigma,  goodcoord,  ncoord,  E0,
                 -leftmultiplier*distancetomove)
                 >
       compressionFatlengthindirection(coord, displacement ,  natoms,
                 epsilon,  sigma,  goodcoord,  ncoord,  E0,
                 -rightmultiplier*distancetomove)
                  )
  {
    leftmultiplier=midmultiplier;
  }
  else
  {
    rightmultiplier=midmultiplier;
  }
 }

} */




addvectormultipliedbyscalor(coord, coord,displacement,
                           movedthisfar, natoms*ncoord);

//printmatrix(displacement,natoms,ncoord);
 distancetomove=movedthisfar;

prevF=currentF;
currentF= compressionfunction( coord,  natoms,
                      goodcoord,  ncoord,
                      epsilon,  sigma,
                      E0);

maxdistancetomove=movedthisfar*12.0;

if (maxdistancetomove>maxmaxdistancetomove)
 {
  maxdistancetomove=maxmaxdistancetomove;
 }

 /*
cout<<"F = "<<currentF
 <<"   W = "<<Wd( coord, natoms, goodcoord, ncoord)
 <<"   E = "<<findenergy(coord, natoms, epsilon,  sigma) - E0
 <<"   maxdistancetomove = "<<maxdistancetomove
 <<"   Fslope = "<< Fslope
```

```
    <<"   distancetomove = "<<distancetomove<<endl;*/

   copyarray(d, prevd,natoms*ncoord);

   prevdellFsquared=Fslope*Fslope;
   iterations++;
   firstiteration=false;


  }while(
(fabs(currentF)>0.0005*natoms)&&(iterations<=maxpossibleiterations)&&(Fslope>
natoms*0.05));

  //cout<<"                                                            Fslope
= "<< Fslope<<endl;
  if (iterations>maxpossibleiterations)
  {cout<<"                                                        Giving up
in compressto3DbyCG after "<<
              iterations<<
              "  iterations."<<endl;
   return 0;
  }
  //cout<<"In correctpositiontochangeenergy iterations = "<<iterations<<endl;
// return;


 return 1;
}


/*This procedure is dimensional compression at constant energy,
done as finding the root of a function (E-Eo)^2+W*/
int  compressto3Dbyfindingroot(double coord[], int natoms,
                        int goodcoord, int ncoord,
                        double epsilon, double sigma)
{
 cout << "                                                         E before
compressto3Dbyfindingroot = "
      << findenergy(coord, natoms, epsilon,  sigma) << endl;
 const double E0=findenergy(coord, natoms, epsilon,  sigma);

 /* Prints an MxN matrix a*/
 // printmatrix(coord, natoms, ncoord);


 const int maxpossibleiterations=1000*natoms;
 /*Because this search is newton-like, it has a possibility
    of not converging and freezing up the programm for that reason
    after a ceratin number of iterations it will just give up.
    */
 double dellF[natoms*ncoord];  //  gradiant of enegrgy

 /*Vars about the function being optimized*/
 double Fslope, currentF, prevF;
 double displacement[natoms*ncoord];

 /*This variable is for constraining the distance
```

```
  Newton's algorithm can jump. It is there, so it
  would not diverge*/
 double maxdistancetomove = 0.1;
 double maxmaxdistancetomove = 0.3*natoms;
 int iterations=0; //number of elapsed iterations

 currentF= compressionfunction( coord,  natoms,
                        goodcoord,  ncoord,
                        epsilon,  sigma,
                        E0);

 double distancetomove=0.1;
 do
  {

   finddellcompressionfunction( coord, natoms,
                       goodcoord,  ncoord,
                       epsilon,  sigma,
                       E0,  dellF);



   Fslope = vectorlength(dellF,natoms*ncoord);

   //cout<<"Fslope = "<< Fslope<<endl;
   copyarray(dellF,displacement,natoms*ncoord);
   //printmatrix(displacement,natoms,ncoord);
   normalizevector(displacement,natoms*ncoord);

   if(distancetomove>maxdistancetomove)
   {distancetomove=maxdistancetomove;}

   double multiplier=30.0;

   do
   {
    multiplier*=0.5;
    /*copies a to b, so b=a*/
    //void copyarray(double a[], double b[], int n)
   }  while(currentF<=compressionFatlengthindirection(coord,  displacement  ,
natoms,
                epsilon,  sigma,  goodcoord,  ncoord,  E0,
                -multiplier*distancetomove));

   {
    /*double bestmultiplier= multiplier;
    double currentmultiplier;

    for
(currentmultiplier=0.0;currentmultiplier<=(multiplier*2.0);currentmultiplier+
=multiplier*0.1)
      {
       if (compressionFatlengthindirection(coord, displacement ,  natoms,
                    epsilon,  sigma,  goodcoord,  ncoord,  E0,
                    -bestmultiplier*distancetomove)
                    >
           compressionFatlengthindirection(coord, displacement ,  natoms,
                    epsilon,  sigma,  goodcoord,  ncoord,  E0,
                    -currentmultiplier*distancetomove)
                     )
                    {bestmultiplier=currentmultiplier;}
      }
    multiplier=bestmultiplier;

    */


    double leftmultiplier=0.0;
    double rightmultiplier=multiplier*2.0;
    double midmultiplier= 0.5*(leftmultiplier+rightmultiplier);

    int i;
    for (i=0; i<15; i++)
    {
     midmultiplier= 0.5*(leftmultiplier+rightmultiplier);
     if (compressionFatlengthindirection(coord, displacement ,  natoms,
                    epsilon,  sigma,  goodcoord,  ncoord,  E0,
                    -leftmultiplier*distancetomove)
                    >
           compressionFatlengthindirection(coord, displacement ,  natoms,
                    epsilon,  sigma,  goodcoord,  ncoord,  E0,
                    -rightmultiplier*distancetomove)
                     )
     {
       leftmultiplier=midmultiplier;
     }
     else
     {
       rightmultiplier=midmultiplier;
     }
    }

   }


   addvectormultipliedbyscalor(coord, coord,displacement,
                             -distancetomove*multiplier, natoms*ncoord);

   //printmatrix(displacement,natoms,ncoord);
    distancetomove*=multiplier;

   prevF=currentF;
   currentF= compressionfunction( coord,  natoms,
                        goodcoord,  ncoord,
                        epsilon,  sigma,
                        E0);

   maxdistancetomove*=multiplier;
   if (prevF<currentF)
   {
    maxdistancetomove*=0.5;

    /*
    addvectormultipliedbyscalor(coord, coord,displacement,
                             0.5*distancetomove, natoms*ncoord);//step back
    currentF= compressionfunction( coord,  natoms,
                         goodcoord,  ncoord,
                         epsilon,  sigma,
                         E0);   */
   }
   else
   {
    maxdistancetomove*=2.0;
    if (maxdistancetomove>maxmaxdistancetomove)
    {
     maxdistancetomove=maxmaxdistancetomove;
    }
   }

   cout<<"F = "<<currentF
    <<"   W = "<<Wd( coord, natoms, goodcoord, ncoord)
    <<"   E = "<<findenergy(coord, natoms, epsilon,  sigma) - E0
    <<"   maxdistancetomove = "<<maxdistancetomove
    <<"   Fslope = "<< Fslope
    <<"   distancetomove = "<<distancetomove<<endl;

   iterations++;
  }while(
(fabs(currentF)>0.001)&&(iterations<=maxpossibleiterations)&&(Fslope>natoms*0
.05));

  cout<<"              Fslope = "<< Fslope<<endl;
  if (iterations>maxpossibleiterations)
  {cout<<"                                                          Giving up
in compressto3Dbyfindingroot after "<<
             iterations<<
             "  iterations."<<endl;
   return 0;
  }
  //cout<<"In correctpositiontochangeenergy iterations = "<<iterations<<endl;
// return;


 return 1;
}








void makecompressionstep(double coord[], int natoms,
                double  epsilon,  double  sigma,  int  goodcoord,  int  ncoord,
double Eo, double steplength)
{
 const int maxsteps=10000; /*for both the first linier steps
                              and the following binary search*/

 double direction[natoms*ndims];
 /*zeroextracoordinates(direction, natoms); */
 double l1=0.0; //closet point
 double l2=0.0; //furthest point
 double lm=0.0; // point in the middle


 double F0 = compressionfunction( coord,  natoms,
                        goodcoord,  ncoord,
                        epsilon,  sigma,
                        Eo);
 finddellcompressionfunction( coord, natoms,
                       goodcoord,  ncoord,
                       epsilon,  sigma,
                       Eo,  direction);
 //findgradient(coord, direction, natoms,epsilon,sigma);
 /*zeroextracoordinates(direction, natoms);overkill*/




 vectormultipliedbyscalor(direction, -1.0, direction, natoms*ncoord);
 //normalizeandmultiplygradient(direction, natoms, -1.0);

 int step=0;

 do    //Loop finds the piece of line to search
 {
  step++;

  l1 = l2;

  l2 = l2+ steplength;
  /*
  cout<<"Eopt="<<energyatlengthindirection(coord, direction, natoms,
                 epsilon,  sigma, l2) <<endl;

  compressionFatlengthindirection(double  coord[],  double  direction[],  int
natoms,
                 double  epsilon,  double  sigma,  int  goodcoord,  int  ncoord,
double Eo,
                 double l)


                 */
 }
 while ((compressionFatlengthindirection(coord, direction, natoms,
                 epsilon,  sigma, goodcoord, ncoord, Eo, l2)       <
        compressionFatlengthindirection(coord, direction, natoms,
                 epsilon,      sigma,   goodcoord,   ncoord,   Eo,   l1))   &&
(step<=maxsteps));


 double F1 = compressionFatlengthindirection(coord, direction, natoms,
                 epsilon,  sigma, goodcoord, ncoord, Eo, l2);



 step=0;
 do  //finds the root
 {
  lm = (l1+l2)*0.5;
  if(
        compressionFatlengthindirection(coord, direction, natoms,
                 epsilon,  sigma, goodcoord, ncoord, Eo, lm)        <
        compressionFatlengthindirection(coord, direction, natoms,
                 epsilon,  sigma, goodcoord, ncoord, Eo, l2) )
  {
   l2=lm;
  }
  else
  { l1=lm;}

  /*
  cout<<"Rootf"<<normalizeddelldotdirection(coord,   direction ,   natoms,
                   epsilon,   sigma, lm )
                   <<"   E="
                   <<energyatlengthindirection(coord, direction, natoms,
                 epsilon,  sigma, lm)
                   <<endl;  */
 }
 while (((l2-l1)>0.0000000000001) && (step<=maxsteps));

 double F2 = compressionFatlengthindirection(coord, direction, natoms,
                 epsilon,  sigma, goodcoord, ncoord, Eo, l2);
 if (F2>F0)
  {
  // cout<<"E1>E0, "<<"E0="<<E0<<" E1=" <<E1 <<" E2="<<E2<<endl;
  }

 addvectormultipliedbyscalor(coord, coord,direction,l2, natoms*ndims) ;

}



int compressto3DbyfindingrootSD(double coord[], int natoms,
                int goodcoord, int ncoord,
                double epsilon, double sigma,
                double steplength,
                double maxgradiant /*When gradient
                smaller than this is reached, the
                search terminates*/
                )
{
 const int minsteps=1;
 int step=0;

 double dell[natoms*ndims];
 cout << "                                                        E before
compressto3DbyfindingrootSD = "
      << findenergy(coord, natoms, epsilon,  sigma) << endl;
 const double E0=findenergy(coord, natoms, epsilon,  sigma);

 do
 {
   step++;

   makecompressionstep(coord, natoms, epsilon, sigma, goodcoord, ncoord, E0,
steplength);
   findgradient(coord, dell, natoms, epsilon, sigma);

   finddellcompressionfunction( coord, natoms,
                        goodcoord,  ncoord,
                        epsilon,  sigma,
                        E0,  dell);
 }
 while((vectorlength(dell,natoms*ndims)>maxgradiant)||(step<minsteps));

 return 1;
}
```

```
/*This procedure is dimensional compression at constant energy,
done as conventional minimization in the direction perpendicular
to the energy gradient.*/
int  optimizeWdatconstE(double coord[], int natoms,
                        int goodcoord, int ncoord,
                        double epsilon, double sigma)
{

 cout << "                                                        E before
optimizeWdatconstE = "
      << findenergy(coord, natoms, epsilon,  sigma) << endl;
 const double originalenergy=findenergy(coord, natoms, epsilon,  sigma);
 const int mininterations = 5;
 const int maxiterations = 30000*natoms;
 double steplength=0.001*sigma;
 double tolerance=0.0;//0.05*sigma;
 double direction[natoms*ncoord];
 double dWd[natoms*ncoord];
 int i;
 double prevE=0.0;
 double E=10e10;
 double prevW=0.0;
 double W=10e10;

 for (i=0; i<maxiterations; i++)
 {
  prevE=E;
  E=findenergy(coord, natoms, epsilon, sigma);
  /*
  if (fabs(E-prevE)>(0.0001*epsilon))
   {steplength/=2;}
  else
   {steplength*=1.1;} */

  /*if (fabs(E-originalenergy)>(0.05 * sigma)) */





  prevW=W;
  W=Wd( coord, natoms, goodcoord, ncoord);
  if ((W<(steplength*10.0)) /*|| (W<tolerance)*/)
  {
   cout << "                                                        E after
optimizeWdatconstE = "
   << findenergy(coord, natoms, epsilon,  sigma) << endl;
   cout << "                                                         Width
is " << W<< endl;
   cout << "                                                        after "
<< i<<" iterations."<< endl;

   return 1;

  }

  /*cout << " E = " << findenergy(coord, natoms, epsilon, sigma)
       << ",  Wd = "<< Wd( coord, natoms, goodcoord, ncoord)
       << ",  steplength = "<<steplength<<endl; */

  dellWdatconstantenergy(coord,  natoms, dWd,
                                   goodcoord,  ncoord,
                                   epsilon,  sigma);
  copyarray(dWd,direction,natoms*ncoord);

  normalizevector(direction,natoms*ncoord);

  //makes result = v1+a*v2
  //void  addvectormultipliedbyscalor(double  result[],  double  v1[],double
v2[],
    //                             double a, int n)

   addvectormultipliedbyscalor(coord, coord,direction,
                                -steplength, natoms*ncoord);
   correctpositiontochangeenergy(coord,natoms,
                                goodcoord,  ncoord,
                                epsilon,  sigma, E, originalenergy);

 }
   cout                                  <<                                 "
optimizeWdatconstE FAILED E = "
   << findenergy(coord, natoms, epsilon,  sigma) << endl;
   cout << "                                                              Width
is " << W<< endl;

 return 0;
}



/*  dell = multiplier*dell/||dell||  */
void   normalizeandmultiplygradient(double   dell[],   int   natoms,   double
multiplier)
{
 int k; //atoms
 int dim;

 double sum=0;
 for ( k = 0;  k < natoms;  k++)
 {
  for (dim=0; dim<ndims; dim++)
  {
     sum+=dell[k*ndims+dim]*dell[k*ndims+dim];
  }
 }
 double length=sqrt(sum);

 for ( k = 0;  k < natoms;  k++)
 {
  for (dim=0; dim<ndims; dim++)
  {
   dell[k*ndims+dim]=dell[k*ndims+dim]/length*multiplier;
  }
 }
 return;
}

/*
double  normalizeddelldotdirection(double  coord[],  double  direction[],  int
natoms,
                 double epsilon, double sigma,
                 double l)
{
  double dell[natoms*3];
  double newcoord[natoms*3];
  addvectormultipliedbyscalor(newcoord, coord,direction,
                                  l, natoms*3);

  findgradient(newcoord, dell, natoms,epsilon,sigma);
  return dotproduct(dell,direction, natoms)
            /vectorlength(direction,3*natoms)
            /vectorlength(dell     ,3*natoms);
} */

/*Returned enegry of the system if it is moved from coordinates coord
in direction direction l units of length.
*/

/*Puts zeros in dimentions not used */
```

```
void zeroextracoordinates(double coord[], int natoms, int maxdims)
{
 int i;  // atom number
 int dim;

 for (i=0;i<natoms;i++)
 {
  for (dim=ndims; dim < maxdims; dim++)
  {coord[i*maxdims+dim]=0.0;}
 }
}

double  energyatlengthindirection(double  coord[],  double  direction[],  int
natoms,
                 double epsilon, double sigma,
                 double l)
{

 double newcoord[natoms*ndims];
 /*Extra array elements are made zero.
 zeroextracoordinates(newcoord, natoms);
 zeroextracoordinates(direction, natoms);*/
 addvectormultipliedbyscalor(newcoord, coord,direction,
                                  l, natoms*ndims);
 return findenergy(newcoord,natoms, epsilon,  sigma) ;
}

void makecgstep(double coord[], int natoms,
                double epsilon, double sigma, double steplength)
{

 const int maxsteps=10000; /*for both the first linier steps
                              and the following binary search*/

 double direction[natoms*ndims];
 /*zeroextracoordinates(direction, natoms); */
 double l1=0.0; //closet point
 double l2=0.0; //furthest point
 double lm=0.0; // point in the middle


 double E0 = findenergy(coord,natoms, epsilon,  sigma) ;

 findgradient(coord, direction, natoms,epsilon,sigma);
 /*zeroextracoordinates(direction, natoms);overkill*/

 normalizeandmultiplygradient(direction, natoms, 1.0);

 int step=0;

 do    //Loop finds the piece of line to search
 {
  step++;

  l1 = l2;

  l2 = l2+ steplength;
  /*
  cout<<"Eopt="<<energyatlengthindirection(coord, direction, natoms,
                epsilon,  sigma, l2) <<endl;  */
 }
 while ((energyatlengthindirection(coord, direction, natoms,
                epsilon,  sigma, l2)        <
       energyatlengthindirection(coord, direction, natoms,
                epsilon,  sigma, l1)) && (step<=maxsteps));
    /*(normalizeddelldotdirection(coord,   direction ,   natoms,
                  epsilon,   sigma, l2)>=0.0); */

 double E1 = energyatlengthindirection(coord, direction, natoms,
                epsilon,  sigma, l2);

 /*if (E1>E0)
  {
   cout<<"E1>E0"<<endl;
  }
 */

 step=0;
 do  //finds the root
 {
  lm = (l1+l2)*0.5;
  if(
       energyatlengthindirection(coord, direction, natoms,
                epsilon,  sigma, lm)        <
       energyatlengthindirection(coord, direction, natoms,
                epsilon,  sigma, l2) )
  {
   l2=lm;
  }
  else
  { l1=lm;}

  /*
  cout<<"Rootf"<<normalizeddelldotdirection(coord,   direction ,   natoms,
                  epsilon,   sigma, lm )
                  <<"   E="
                  <<energyatlengthindirection(coord, direction, natoms,
                epsilon,  sigma, lm)
                  <<endl;  */
 }
 while (((l2-l1)>0.0000000000001) && (step<=maxsteps));

 double E2 = energyatlengthindirection(coord, direction, natoms,
                epsilon,  sigma, l2);
 if (E2>E0)
  {
  // cout<<"E1>E0, "<<"E0="<<E0<<" E1=" <<E1 <<" E2="<<E2<<endl;
  }

 addvectormultipliedbyscalor(coord, coord,direction,l2, natoms*ndims) ;

}

double optimizeEonline(double coord[], double direction[],int natoms,
                        int ncoord,
                       double epsilon, double sigma,
                       double originaldistance, int maxcycles)
{
    double movedthisfar;
    double leftmultiplier=0.0;
    double rightmultiplier=originaldistance;
    double midmultiplier= 0.5*(leftmultiplier+rightmultiplier);

    int i;
    for (i=0; i<maxcycles; i++)
    {
     midmultiplier= 0.5*(leftmultiplier+rightmultiplier);
     if (energyatlengthindirection(coord, direction, natoms,
                 epsilon,  sigma,
                     leftmultiplier)
                     >
         energyatlengthindirection(coord, direction, natoms,
                 epsilon,  sigma,
                     rightmultiplier)
                      )
     {
       leftmultiplier=midmultiplier;
       movedthisfar=midmultiplier; //cout<<"moving right"<<endl;
     }
     else
     {
       rightmultiplier=midmultiplier;
       movedthisfar=midmultiplier;
       //cout<<"moving left"<<endl;
     }
    }
  //cout<<ncoord<<endl<<flush;
  return movedthisfar;
}

double optimizeEonline2(double coord[], double direction[],int natoms,
                        int ncoord,
                       double epsilon, double sigma,
                       double originaldistance, int maxcycles)
{
    double movedthisfar;
    double leftmultiplier=0.0;
    double rightmultiplier=originaldistance;
    double midmultiplier= 0.5*(leftmultiplier+rightmultiplier);

    double dell[ncoord];
    find_gradientE(coord, dell, natoms, epsilon, sigma);
    if (dotproduct(direction, dell, ncoord)>0.0)
    {
     vectormultipliedbyscalor(direction, -1.0,
                                  direction, ncoord);
    }
    /*
    int j;
     dotproduct(double v1[], double v2[], int n)
     void find_gradientE(double coord[], double dell[], int natoms,
                  double epsilon, double sigma);

      */

    int i;
    for (i=0; i<maxcycles; i++)
    {
     midmultiplier= 0.5*(leftmultiplier+rightmultiplier);
     if (energyatlengthindirection(coord, direction, natoms,
                 epsilon,  sigma,
                     leftmultiplier)
                     >
         energyatlengthindirection(coord, direction, natoms,
                 epsilon,  sigma,
                     rightmultiplier)
                      )
     {
       leftmultiplier=midmultiplier;
       movedthisfar=midmultiplier; //cout<<"moving right"<<endl;
     }
     else
     {
       rightmultiplier=midmultiplier;
       movedthisfar=midmultiplier;
       //cout<<"moving left"<<endl;
     }
    }

  return movedthisfar;
}


double optimizeEonlineSafe(double coord[],int natoms,
                         int ncoord,
                        double epsilon, double sigma,
                        double originaldistance, int maxcycles)
{
    double direction[ncoord];
    double movedthisfar;
    double leftmultiplier=0.0;
    double rightmultiplier=originaldistance;
    double midmultiplier= 0.5*(leftmultiplier+rightmultiplier);


    findgradient(coord, direction, natoms, epsilon, sigma);
    double longestcomponent=vectorlongestcompont(direction, ncoord);

    vectormultipliedbyscalor(direction, (1.0/longestcomponent),
                                  direction, ncoord);

    int i;
    for (i=0; i<maxcycles; i++)
    {
     midmultiplier= 0.5*(leftmultiplier+rightmultiplier);
     if (energyatlengthindirection(coord, direction, natoms,
                 epsilon,  sigma,
                     leftmultiplier)
```

```
                     >
         energyatlengthindirection(coord, direction, natoms,
                 epsilon,  sigma,
                     rightmultiplier)
                      )
     {
       leftmultiplier=midmultiplier;
       movedthisfar=midmultiplier; //cout<<"moving right"<<endl;
     }
     else
     {
       rightmultiplier=midmultiplier;
       movedthisfar=midmultiplier;
       //cout<<"moving left"<<endl;
     }
    }
  //cout<<ncoord<<endl<<flush;
  addvectormultipliedbyscalor(coord, coord, direction,
                                  movedthisfar, ncoord);

  return movedthisfar;
}


void sdoptSafe(double coord[], int natoms,
                double epsilon, double sigma,
                double steplength,
                double maxgradiant /*When gradient
                smaller than this is reached, the
                search terminates*/)
{
 const int minsteps=1;
 int step=0;
 int ncoord= natoms*ndims;
 double dell[natoms*ndims];
 /*zeroextracoordinates(dell, natoms);*/
 double originaldistance=sigma;


 do
 {
   step++;
   originaldistance=optimizeEonlineSafe( coord, natoms,
                          ncoord,
                         epsilon,  sigma,
                         originaldistance, 5);
   /*optimizeEonline(double coord[], double direction[],int natoms,
                         int ncoord,
                        double epsilon, double sigma,
                        double originaldistance, int maxcycles) */
//   makecgstep(coord, natoms, epsilon, sigma, steplength);
   findgradient(coord, dell, natoms, epsilon, sigma);
//   cout<<"gradient"<<vectorlength(dell,natoms*ndims)<<
//    ",   E"<<findenergy(coord,natoms, epsilon,  sigma)<<endl ;

 }
 while((vectorlength(dell,natoms*ndims)>maxgradiant)||(step<minsteps));
```

```
}

void sdopt(double coord[], int natoms,
                double epsilon, double sigma,
                double steplength,
                double maxgradiant /*When gradient
                smaller than this is reached, the
                search terminates*/)
{
 const int minsteps=1;
 int step=0;

 double dell[natoms*ndims];
 /*zeroextracoordinates(dell, natoms);*/



 do
 {
   step++;

   makecgstep(coord, natoms, epsilon, sigma, steplength);
   findgradient(coord, dell, natoms, epsilon, sigma);
//   cout<<"gradient"<<vectorlength(dell,natoms*ndims)<<
//    ",   E"<<findenergy(coord,natoms, epsilon,  sigma)<<endl ;

 }
 while((vectorlength(dell,natoms*ndims)>maxgradiant)||(step<minsteps));


}



void oldcgopt(double coord[], int natoms,
                double epsilon, double sigma,
                double steplength,
                double maxgradiant /*When gradient
                smaller than this is reached, the
                search terminates*/)

{
 const int minsteps=1;
 int step=0;
 int ncoord=natoms*ndims;
 double displacement;//how far to move in each step;
 double dellE[ncoord];
 double normalizeddellE[ncoord];
 /*zeroextracoordinates(dell, natoms);*/
 double prevdellElength=1e30;
 double Direction[ncoord];
 double prevDirection[ncoord];
 zeroarray(prevDirection, ncoord);
 double beta=0.0;
 double distancetosample=natoms*0.1;
 find_gradientE(coord, dellE, natoms, epsilon, sigma);

 do
 {
   step++;

  //cout<<1<<endl<<flush;


   if (step>1e20)
   {
    //cout<<2<<endl<<flush;

    beta=vectorlength(dellE,ncoord)/prevdellElength;
    prevdellElength=(vectorlength(dellE,ncoord));
    //makes result = a*v1+b*v2
    addvectorsmultipliedbyscalors(Direction, dellE, Direction,
                              -1.0/prevdellElength,     beta*displacement,
ncoord);
   }
   else
   {
     //cout<<3<<endl<<flush;dsdf1hgsdh1ffghGHGHGGGGGGHGGHGHGaaaAAA
     //makes result = a*v
    vectormultipliedbyscalor(Direction, -1.0,
                              dellE, ncoord);
    //copyarray(Direction,dellE,ncoord);

    normalizevector(Direction,ncoord);
   }

   prevdellElength=(vectorlength(dellE,ncoord));
   //cout<<4<<endl<<flush;

   //find displacement here
   displacement=optimizeEonline(coord,  Direction, natoms,
                          ndims,
                         epsilon,  sigma,
                         distancetosample,  50);
   distancetosample=displacement*9.0;

   //cout<<5<<endl<<flush;

   addvectormultipliedbyscalor(coord, coord, Direction,displacement, ncoord);

   //cout<<6<<endl<<flush;

   copyarray(Direction,prevDirection, ncoord);

   //cout<<7<<endl<<flush;

   find_gradientE(coord, dellE, natoms, epsilon, sigma);

   //cout<<8<<endl<<flush;

   cout<<"gradient "<<vectorlength(dellE,ncoord)<<
   ",   E "<<findenergy(coord,natoms, epsilon,  sigma)<<
   ",   l "<<distancetosample<<endl<<flush ;

  //cout<<9<<endl<<flush;
 }
 while((vectorlength(dellE,ncoord)>maxgradiant)||(step<minsteps));
cout << step << " steps"<<endl<<flush;
}

void cgopt(double coord[], int natoms,
                double epsilon, double sigma,
                double steplength,
                double maxgradiant /*When gradient
                smaller than this is reached, the
                search terminates*/)

{
 const int ITMAX=20000;

 int ncoord=natoms*ndims;
 int j, its, iter;
 double gg, gam, fp, dgg, fret;
 double g[ncoord], h[ncoord], x[ncoord], xi[ncoord];
 int n=ncoord;
 double distancetosample=natoms*0.1;
 double movedthisfar;

 fp=findenergy(coord,natoms, epsilon,  sigma);
 find_gradientE(coord, xi, natoms, epsilon, sigma);

 for (j=0;j<ncoord;j++)
 {
  g[j]= -xi[j];
  xi[j] = h[j] = g[j];
 }

 for (its=0; its<ITMAX; its++)
 {
  iter=its;
  {
   movedthisfar=optimizeEonline2(coord, xi, natoms,
                       ncoord,
                      epsilon,  sigma,
                     distancetosample, 10);

   distancetosample=movedthisfar*10.0;
   addvectormultipliedbyscalor(coord, coord, xi, movedthisfar, ncoord);
   for (j=0;j<ncoord;j++)
   {
    xi[j]*=movedthisfar;
   }
   fret=findenergy(coord,natoms, epsilon,  sigma);
  }

  fp=fret;
  find_gradientE(coord, xi, natoms, epsilon, sigma);
  /*cout<<"gradient "<<vectorlength(xi,ncoord)<<
```

```
    ",   E "<<findenergy(coord,natoms, epsilon,  sigma)<<
    ",   l "<<distancetosample<<endl<<flush ; */
  if (vectorlength(xi,ncoord)<maxgradiant)
   return;

  dgg=gg=0.0;
  for (j=0;j<ncoord;j++)
   {
    gg+=g[j]*g[j];
    //dgg+=xi[j]*xi[j]; //Fletcher-Reeves
    dgg+=(xi[j]+g[j])*xi[j]; //Polak-Ribiere
   }

  if (gg==0.0)
   return;
  gam=dgg/gg;
  for (j=0;j<n;j++)
  {
   g[j]= -xi[j];
   xi[j]=h[j]=g[j]+gam*h[j];
  }
 }
 return;
}

/*
void   findnewtondisplacement(double   coords[],   double   displacement[],int
natoms,
                  double epsilon, double sigma)
{
 double dell[natoms*ndims];
 findgradient(coords, dell, natoms, epsilon, sigma);
 zeroextracoordinates(dell, natoms);
 int k;
 for ( k = 0;  k < natoms*3;  k++)
 {
     displacement[k]=-coords[k]/dell[k];
 }
}



void makenewtonstep(double coord[], int natoms,
                double epsilon, double sigma, double steplength)
{
 double displacement[3*natoms];
 findnewtondisplacement(coord, displacement, natoms, epsilon, sigma);
 if (vectorlength(displacement,natoms*3)>steplength)
 {normalizeandmultiplygradient(displacement, natoms, steplength);}
 int k;
 for ( k = 0;  k < natoms*3;  k++)
 {
     coord[k]=coord[k]-displacement[k];
 }


 return;
```

```
} */


/*Returnes the distance of atom atomnumber to the closest atom.
   Pre:  natoms > 1      */
double mindistancetootheratoms(double coord[], int natoms,
                                    int goodcoord,
                                    int ncoord,
                                    int atomnumber)
{
    // atom to which the distance is being measured
    int atom;
    //We will return this. Now it is set to a very high value
    double mindistance=1e20;
    for(atom=0;atom<natoms;atom++)
    {
     if (atomnumber!=atom)
     /*The distance to itself is 0.
       We do not want that for an answer.*/
     {
       if (dist(coord, atomnumber, atom)<mindistance)
       {
         mindistance=dist(coord, atomnumber, atom);
       }
     }
     //cm3d[dim]=cm3d[dim]+coord[ncoord*atom+dim];
    }

    return mindistance;
}

bool atomic_crashes_exist(double coord[],int natoms,int ndims,
                        double sigma)

{
  bool squizzedatomsexist=false;
  double squizzeddistance=0.50*sigma;
  int atom;
  for(atom=0;atom<natoms;atom++)
  {
   if (squizzeddistance>mindistancetootheratoms(coord, natoms,
                                    ndims,
                                    ndims,
                                    atom))
   {
    squizzedatomsexist=true;
   }
  }
  return squizzedatomsexist;
}


/*Does one tunneling step through extra dimensions
returnes true if successful.*/

/*The glitch is here*/
int oneNDtravel(double coord3[],int natoms,
                                    int goodcoord, int ncoord,
                                    double epsilon, double sigma)
{
 const double faluremultiplier = 1.0;
 const double successmultiplier = 1.0;


 double coord3backup[natoms*goodcoord];

 copyarray(coord3,coord3backup,natoms*goodcoord);

 double originalenergy=findenergy(coord3backup,natoms, epsilon,  sigma);

 double coordN[natoms*ncoord];

 go3toN(coord3, coordN,  natoms, goodcoord,  ncoord);

 travelND(coordN,  natoms,
               goodcoord,  ncoord,
               epsilon,  sigma);


 bool foundsomeminimum = false;

 double E=findenergy(coordN, natoms, epsilon,  sigma);
 if ((-1e5<E)&&(E<0))
 {
  /*foundsomeminimum= optimizeWdatconstE(
                       coordN, natoms, goodcoord,  ncoord,
                         epsilon,  sigma); */

  foundsomeminimum = compressto3DbyTSwalk(
                       coordN, natoms, goodcoord,  ncoord,epsilon,  sigma);

  /*foundsomeminimum = cgoptWDatConstantE( coordN, natoms,
                 epsilon,  sigma,
                 goodcoord,  ncoord,
                sigma,
                0.001 When gradient
                smaller than this is reached, the
                search terminates);*/



  /*  foundsomeminimum = compressto3Dbyfindingroot(
                       coordN, natoms, goodcoord,  ncoord,epsilon,  sigma);

  foundsomeminimum = compressto3DbyfindingrootSD(
                       coordN,  natoms,  goodcoord,   ncoord,epsilon,   sigma,
0.02*natoms,1e-1*natoms
                         ); */
  //outputatoms(coordN,natoms,epsilon, sigma)  ;


  if (foundsomeminimum)
  {
   goNto3( coordN,  coord3,  natoms, goodcoord,  ncoord);

   bool                        no_atomic_crashes=!atomic_crashes_exist(coord3,
natoms,goodcoord,sigma);

   if (no_atomic_crashes)
   {
    cgopt(  coord3,natoms,epsilon, sigma, 0.0002*natoms,1e-3*natoms);
   }


   if (no_atomic_crashes &&
       ((originalenergy+0.03*epsilon)<
       findenergy(coord3,natoms, epsilon,  sigma)))
   {

    foundsomeminimum=0;
    cout<<"
Minimized to high E = ";
    cout<<findenergy(coord3, natoms, epsilon,  sigma)<<endl;
    if (options.reporiting_depth>4)
    {outputatoms(coord3,natoms,epsilon, sigma);}
    copyarray(coord3backup,coord3,natoms*goodcoord);
    options.travel_step_size*=faluremultiplier;
   }


   //outputatoms(coord3,natoms,epsilon, sigma)  ;
  }
  else
  {
   cout<<"                                                                   Ran
into multidimensional local minimum"<<endl;
   options.travel_step_size*=faluremultiplier;
  }
 }
 else
 {
    cout<<"                                                                  Ran
into a crazy place on PES, no compression done";
    cout<<endl;
    options.travel_step_size*=faluremultiplier;
 }
 ndims=goodcoord;

 E=findenergy(coord3, natoms, epsilon,  sigma);
 if ((-1e5<E)&&(E<0.0))
 {
  options.travel_step_size*=successmultiplier;
 }
 else
 {
  copyarray(coord3backup,coord3,natoms*goodcoord);
 }
 return foundsomeminimum;
}
```

```
double known_global_minimum(int natoms)
{
 switch (natoms)
 {
  case 1: return (0.0+0.001); break;
  case 2: return (-1.0+0.001); break;
  case 3: return (-3.0+0.001); break;
  case 4: return (-6.0+0.001); break;
  case 5: return -9.10385; break;
  case 6: return -12.71206; break;
  case 7: return -16.50538; break;
  case 8: return -19.82148; break;
  case 9: return -24.1133; break;
  case 10: return -28.42253; break;
  case 11: return -32.7659; break;
  case 12: return -37.967; break;
  case 13: return -44.32680; break;
  case 14: return -47.84515; break;
  case 15: return -52.32262; break;
  case 16: return -56.815; break;
  case 17: return -61.317; break;
  case 18: return -66.530; break;
  case 19: return -72.659; break;
  case 20: return -77.177; break;
  case 21: return -81.684; break;
  case 22: return -86.809; break;
  case 23: return -92.844; break;
  case 24: return -97.348; break;
  case 25: return -102.372; break;
  case 26: return -108.315; break;
  case 27: return -112.873; break;
  case 28: return -117.822; break;
  case 29: return -123.587; break;
  case 30: return -128.286; break;
  case 31: return -133.586; break;
  case 32: return -139.635; break;
  case 33: return -144.842; break;
  case 34: return -150.044; break;
  case 35: return -155.756; break;
  case 36: return -161.825; break;
  case 37: return -167.033; break;
  case 38: return -173.928; break;
  case 39: return -180.03318; break;
  case 40: return -185.24983; break;
  case 41: return -190.53627; break;
  case 42: return -196.27753; break;
  case 43: return -202.36466; break;
  case 44: return -207.68872; break;
  case 45: return -213.78486; break;
  case 46: return -220.6803; break;
  case 47: return -226.01225; break;
  case 48: return -232.19952; break;
  case 49: return -239.09186; break;
  case 50: return -244.54992; break;
  case 51: return -251.25396; break;
  case 52: return -258.22999; break;
  case 53: return -265.20301; break;
  case 54: return -272.20863; break;
  case 55: return -279.2484; break;
  case 56: return -283.64310; break;
  case 57: return -288.34262; break;
  case 58: return -294.37814; break;
  case 59: return -299.7380; break;
  case 60: return -305.87547; break;
  case 61: return -312.00889; break;
  case 62: return -317.35390; break;
  case 63: return -323.48973; break;
  case 64: return -329.62014; break;
  case 65: return -334.97153; break;
  case 66: return -341.11059; break;
  case 67: return -347.25200; break;
  case 68: return -353.39454; break;
  case 69: return -359.88256; break;
  case 70: return -366.89225; break;
  case 71: return -373.34966; break;
  case 72: return -378.63725; break;
  case 73: return -384.78937; break;
  case 74: return -390.908; break;
  case 75: return -397.49233; break;
  case 76: return -402.89486; break;
  case 77: return -409.08351; break;
  case 78: return -414.79440; break;
  case 79: return -421.81089; break;
  case 80: return -428.08356; break;
  case 81: return -434.34364; break;
  case 82: return -440.55042; break;
  case 83: return -446.92409; break;
  case 84: return -452.65721; break;
  case 85: return -459.05579; break;
  case 86: return -465.38449; break;
  case 87: return -472.09816; break;
  case 88: return -479.0326; break;
  case 89: return -486.05391; break;
  case 90: return -492.43390; break;
  case 91: return -498.8110; break;
  case 92: return -505.18530; break;
  case 93: return -510.87768; break;
  case 94: return -517.26413; break;
  case 95: return -523.64021; break;
  case 96: return -529.87914; break;
  case 97: return -536.68138; break;
  case 98: return -543.66536; break;
  case 99: return -550.66652; break;
  case 100: return -557.0398; break;
  case 101: return -563.41130; break;
  case 102: return -569.36365; break;
  case 103: return -575.76613; break;
  case 104: return -582.08664; break;
  case 105: return -588.26650; break;
  case 106: return -595.0610; break;
  case 107: return -602.0071; break;
  case 108: return -609.03301; break;
  case 109: return -615.4111; break;
  case 110: return -621.7882; break;
```

```
  case 111: return -628.0684; break;
  case 112: return -634.8746; break;
  case 113: return -641.7947; break;
  case 114: return -648.833; break;
  case 115: return -655.7563; break;
  case 116: return -662.8093; break;
  case 117: return -668.2827; break;
  case 118: return -674.7696; break;
  case 119: return -681.4191; break;
  case 120: return -687.0219; break;
  case 121: return -693.8195; break;
  case 122: return -700.9393; break;
  case 123: return -707.8021; break;
  case 124: return -714.9208; break;
  case 125: return -721.3032; break;
  case 126: return -727.3498; break;
  case 127: return -734.4796; break;
  case 128: return -741.332; break;
  case 129: return -748.4606; break;
  case 130: return -755.2710; break;
  case 131: return -762.4415; break;
  case 132: return -768.0422; break;
  case 133: return -775.0232; break;
  case 134: return -782.2061; break;
  case 135: return -790.2781; break;
  case 136: return -797.4532; break;
  case 137: return -804.6314; break;
  case 138: return -811.812; break;
  case 139: return -818.9938; break;
  case 140: return -826.1746; break;
  case 141: return -833.3585; break;
  case 142: return -840.5386; break;
  case 143: return -847.7216; break;
  case 144: return -854.9044; break;
  case 145: return -862.0870; break;
  case 146: return -869.2725; break;
  case 147: return -876.4612; break;
  case 148: return -881.0729; break;
  case 149: return -886.6934; break;
  case 150: return -893.3102; break;
 }
}

void centeratomsin3d(double coord[], int natoms,
                                    int goodcoord,
                                    int ncoord)
{
 int atom;
 int dim;
 double cm3d[goodcoord];
 for(dim=0;dim<goodcoord;dim++)
 {
   cm3d[dim]=0.0;

   //Find the center of mass in this dimension
   for(atom=0;atom<natoms;atom++)
    {
     cm3d[dim]=cm3d[dim]+coord[ncoord*atom+dim];
    }
   cm3d[dim]=cm3d[dim]/natoms;

   //Center all atoms
   for(atom=0;atom<natoms;atom++)
    {
     coord[ncoord*atom+dim]-=cm3d[dim];
    }
 }


 return;
}




/*
 Sets the initial position of all atoms an
 approximatly spherical formation.
*/
void initrandomdata(double coord[],int natoms,int ndims,
                       double epsilon, double sigma)
{
 int atom,dim;

 /*The whole point of using this variable is to make sure atoms
 only go into a phere, not into a cube, inscribed around it.*/
 double sum;


 for (atom=0; atom<natoms; atom++)
 {
  //This loop initializes one atomic position
  do
  {
   sum=0.0;
   for (dim=0; dim<ndims; dim++)
   {
    coord[atom*ndims+dim]=1.0-2.0*((double)rand())/DIVISOR;
    sum+=coord[atom*ndims+dim]*coord[atom*ndims+dim];
   }
  } while (sum>1);

 }

  cout<< "atomic positions initialized at random"<<endl;
  correctpositiontochangeenergy(coord,  natoms,
                                   ndims,ndims,
                                   epsilon,  sigma,
                                  findenergy(coord,     natoms,     epsilon,
sigma),/*current energy*/
                                  0.0/*the energy we want*/);
  //optimizes the cluser
  //cgopt(  coord,natoms,epsilon, sigma, 0.2,1e-3);
  /*compressto3Dbyfindingroot(coord, natoms,
                         ndims, ndims,
                         epsilon, sigma) */
  cout<< "energy has been driven to 0"<<endl;
  //sdoptSafe(  coord,natoms,epsilon, sigma, 2.0,1e-2);
  sdopt(  coord,natoms,epsilon, sigma, 0.2,1e-3);


  cout<< "optimization done"<<endl;
  /*
  Optimization of a random cluster often leads to
  strayaway atoms. This block of code checks for
  that.
  */
  bool strayatomsexist=false;
  double straydistance=2.0*sigma;
  for(atom=0;atom<natoms;atom++)
  {
   if (straydistance<mindistancetootheratoms(coord, natoms,
                                    ndims,
                                    ndims,
                                    atom))
   {
    strayatomsexist=true;
   }
  }

  /*If stray atoms were found in the optimized cluster,
  the whole procedure has to be repeated again.*/
  if (strayatomsexist)
  {
   initrandomdata( coord, natoms, ndims,
                        epsilon,  sigma);
  }

  else         if         (findenergy(coord,         natoms,         epsilon,
sigma)<=known_global_minimum(natoms))
  {
   cout<<"
Accidentally initialized to global minimum: Reinitializing"<<endl;
   initrandomdata( coord, natoms, ndims,
                        epsilon,  sigma);
  }
  return;


 cgopt(coord,  natoms, epsilon, sigma,
               0.2,
               0.01/*When gradient
               smaller than this is reached, the
               search terminates*/);
 return;
}

void skiptonextline(ifstream &ifs)
{
  char buffer[255];
  ifs.getline(buffer,255);
}


void set_known_global_minimum_energy()
{
 if(options.termination_energy==0)
 {
  options.termination_energy=known_global_minimum(options.natoms);
 }
 return;
}

int main(int argc, char *argv[])
{
 // cout<<INT_MAX<<endl;



 ifstream ifs(&(*argv[1]));
 if(!ifs)
 {
  cerr<<"Error: unable to open file" << endl;
  return 1;
 };

 ifs>>options.comment;
 skiptonextline(ifs);

 ifs>>options.natoms;
 skiptonextline(ifs);


 ifs>>options.goodcoord;
 skiptonextline(ifs);

 ifs>> options.epsilon>> options.sigma;
 skiptonextline(ifs);
 //cout<<a<<"   "<<b<<endl;   //debug

 int n=options.goodcoord*options.natoms;
 /*Vector containing the coordinates
 of all LJ atoms positions 0,1,2 contain
 x1, y1, z1 etc.
 */
 double coord[n];
 zeroarray(coord, n);


  ifs>>options.dimsfortravel;
  skiptonextline(ifs);
  ifs>>options.dimsforquench;
  skiptonextline(ifs);
  ifs>>options.way_to_optimize;
  skiptonextline(ifs);
  ifs>>options.way_to_travel;
  skiptonextline(ifs);
  ifs>>options.travel_length_distribution;
  skiptonextline(ifs);
  ifs>>options.travel_step_size;
  skiptonextline(ifs);
  ifs>>options.travel_step_unit;// 1*,  sqrt(N)*, or N*
  skiptonextline(ifs);
  ifs>>options.number_of_travel_steps;
  skiptonextline(ifs);
  ifs>>options.way_to_quench;
  skiptonextline(ifs);
  ifs>>options.quench_step_size;
  skiptonextline(ifs);
  ifs>>options.quench_step_unit;// 1*,  sqrt(N)*, or N*
  skiptonextline(ifs);
  ifs>>options.number_of_quench_steps;
  skiptonextline(ifs);
  ifs>>options.quench_tolerance;
  skiptonextline(ifs);
  ifs>>options.quench_tolerance_unit;// 1*,  sqrt(N)*, or N*
  skiptonextline(ifs);
  ifs>>options.termination_energy;
  skiptonextline(ifs);
  ifs>>options.reporiting_depth;
  skiptonextline(ifs);
  ifs>>options.random_seed; //0 makes the system use time
  skiptonextline(ifs);

 ndims=options.goodcoord;
 int linecounter;

 for ( linecounter = 0;  linecounter < options.natoms;  linecounter++)
     {

         ifs>>coord[linecounter*ndims]>>coord[linecounter*ndims+1]
             >>coord[linecounter*ndims+2];
          skiptonextline(ifs);
  }
 ifs.close();

 set_known_global_minimum_energy();

 long randomseedtime=time(NULL);

 if ((options.random_seed==0)||(options.random_seed==-1))
 {
  srand ( randomseedtime );
  cout <<"Random seed set to "<<randomseedtime<<endl;
  if (options.random_seed==-1)
   {

    initrandomdata(coord,options.natoms, options.goodcoord,
                        options.epsilon, options.sigma);

    outputatoms(coord,options.natoms,options.epsilon, options.sigma);
    cout<<endl;
   }
 }
 else
 {srand ( options.random_seed );}

 //ndims=1;
 //cgopt(  coord,options.natoms,options.epsilon, options.sigma, 0.2,1e-3);
 sdopt(  coord,options.natoms,options.epsilon, options.sigma, 0.2,1e-3);
 /*optimizeWdatconstE( coord,  natoms,
                        2,  3,
                        epsilon,  sigma) ;*/

 outputatoms(coord,options.natoms,options.epsilon, options.sigma);

 long runnumber;

 double currentenergy, previousenergy;
 currentenergy=findenergy( coord, options.natoms,
       options.epsilon, options.sigma);
 previousenergy=currentenergy;

 for (runnumber=1; runnumber<=1000000; runnumber++)
 {
  oneNDtravel(        coord,        options.natoms,        options.goodcoord,
options.dimsfortravel,options.epsilon,  options.sigma);

  currentenergy=findenergy( coord, options.natoms,
       options.epsilon, options.sigma);


  if ((options.reporiting_depth>3) ||
    (fabs(currentenergy-previousenergy)>0.05*options.epsilon))
  {outputatoms(coord,options.natoms,options.epsilon, options.sigma);}

  previousenergy=currentenergy;

  cout <<runnumber<< "     "<< currentenergy <<endl;

  //when the known global minimum is reached, the program stops.
  if (currentenergy<=options.termination_energy)
  {return 0;}
 }
  //outputatoms(coord,natoms,epsilon, sigma);
 return 0;
}
```

APPENDIX B

An Alternative Way for CO Ligation of Metallocycle 129c1

In the carbonyl adsorption step of the Mo-promoted reaction metallocycle **159c3** could form instead of **159c2**. Although there is no energy preference for the formation of one of these intermediates over the other, **159c3** does not lead to a very different reaction path because it fails to transfer the carbonyl group into the metallocycle. Instead it rearranges into **159c2** after overcoming a 1.9 kcal/mol free energy barrier.

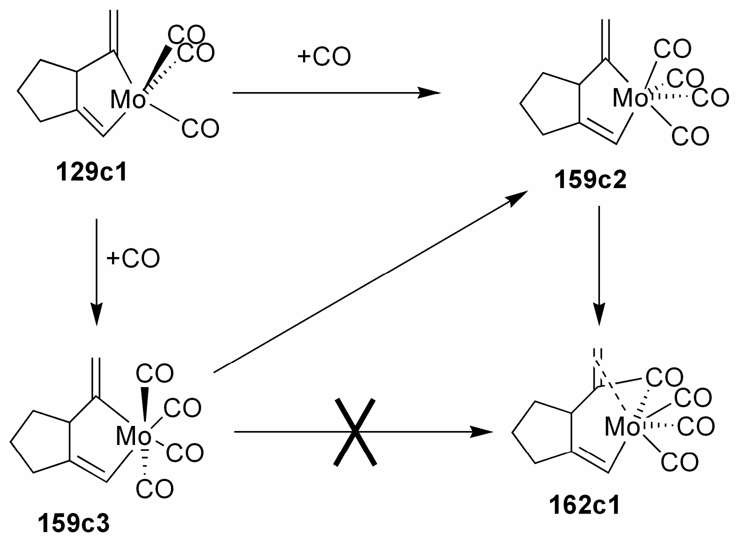


Figure 73 Two ways intermediate 129c1 can adsorb a CO.

APPENDIX C

A Mechanism for the Mo-Promoted Reaction Where There Is No CO Adsorption Immediately Following the Oxidative Addition Step

Because adding a CO from the media to metallocycle **129c1** required overcoming a 8.6 kcal/mol free energy barrier, we considered a pathway where CO insertion immediately followed the oxidative addition step.

Starting with metallocycle **129c1** and using coordinate-driven methods, we found transition states leading from that minimum to metallocycles **155c2** and **155c3** (**Figure 74**). The height of the free energy barrier to get to **155c2** was 32.4 kcal/mol relative to **131c3**. The barrier that led to **155c3** was slightly higher, 34.0 kcal/mol.

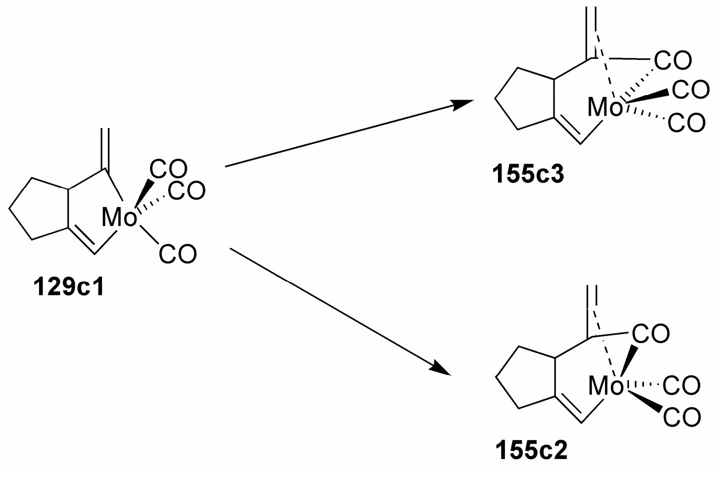


Figure 74 CO insertion into the metallocycle in an alternative mechanism of the Mo-promoted reaction.

Because the barrier that led to **155c2** was slightly lower in free energy than the barrier that led to **155c3**, we decided to explore what happens after the formation of metallocycle **155c2**.

Therefore attempts were made to find the transition state between **155c2** and **157c1**. However, we discovered that this was a two-step process. The first step required only 2.9 kcal/mol of free energy. It did not change the connectivity but only the geometry around the molybdenum center leading to the intermediate **155c4** (**Figure 75**). This was essentially a rearrangement of trigonal bipyramidal molybdenum center to octahedral geometry. The free energy dropped from 20.0 kcal/mol to 14.4 kcal/mol relative to **131c3**.

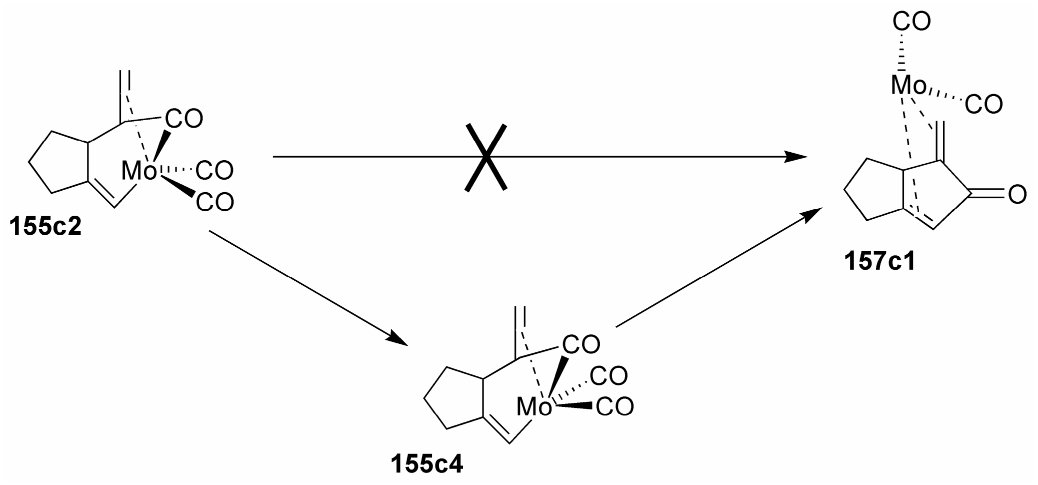


Figure 75 The two-step conversion of 155c2 to 157c1.

In the next step, also shown in **Figure 75**, molybdenum is reductively eliminated from the ring system **155c4** via a transition state which is only 3.9 kcal/mol above metallocycle **155c4** to form **157c1**.

Alternatively, instead of rearranging, **155c2** could react with a carbon monoxide. The transition state for this reaction was not located, because placement of a carbon monoxide molecule in the vicinity of **155c2** followed by an energy minimization led to the formation of **156c1**, with no transition state.

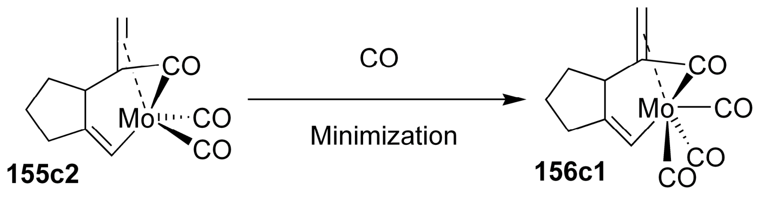


Figure 76 CO adsorption by 155c2.

We mapped parts of several pathways of the Mo-promoted reaction in which CO does not attach to the metal center immediately following the oxidative addition step. All of these are shown in **Figure 77**. They required overcoming free energy barriers much higher than the barrier between minima **129c1** and **159c2**, so they do not contribute significantly to the reaction.

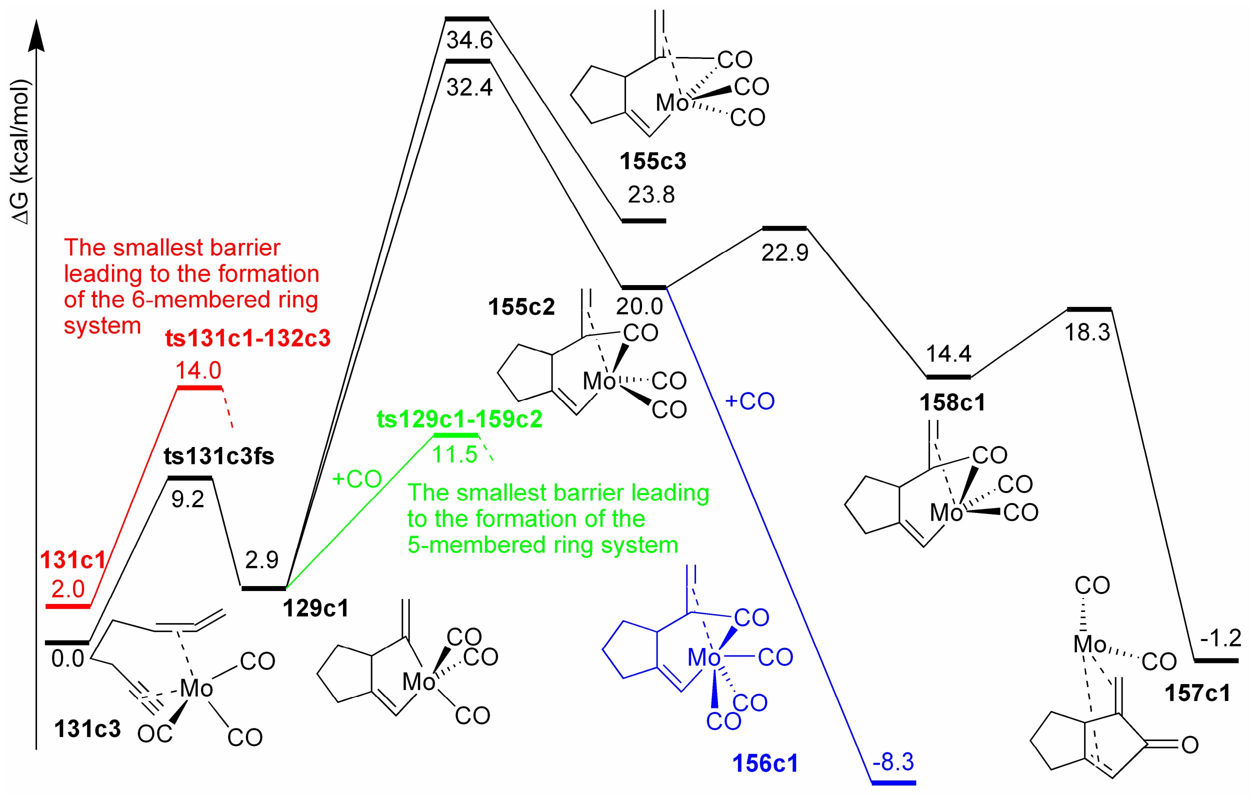


Figure 77 Mo-promoted cyclocarbonylation mechanisms, where CO does not add to the Mo center immediately following the oxidative addition.

Parts of pathways where there is an extra CO present are shown in blue and green.

APPENDIX E

Mechanisms of the Rh-Catalyzed Reaction Where the Next Step After the Oxidative Addition Is Not Ligation of Rh By a CO

Starting with **127c6**, a pathway was determined for the remaining steps of the rhodium-catalyzed reaction: CO insertion into the ring system and reductive elimination of rhodium. Insertion of CO *cis* to C1 between Rh and C1 to form **140c1** had a free energy barrier of 9.5 kcal/mol. Insertion of CO *trans* to C1 between Rh and C8 that led to the formation of **142c1** had a much higher free energy barrier, 26 kcal/mol. Both of these processes are shown in **Figure 78**. The large energy difference between the transition states leading to **140c1** and **142c1** was attributed to the carbonyl being in conjugation with the diene in **140c1**.

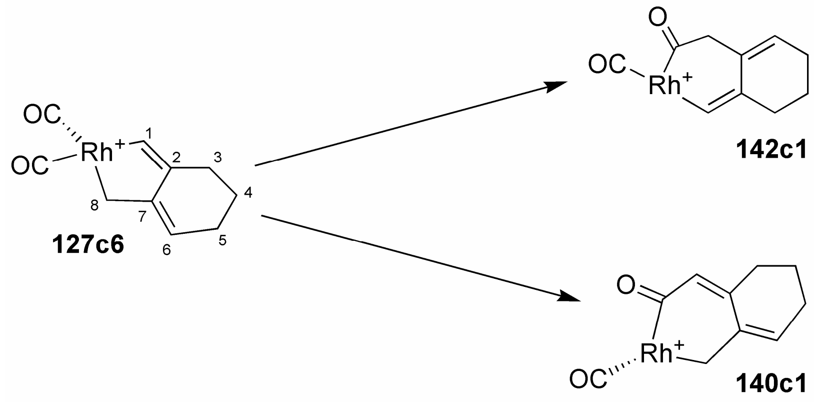


Figure 78 An alternative way to insert a CO into the metallocycle.

Finally reductive elimination of rhodium from **140**, shown in **Figure 79**, was found to require overcoming a 36.4 kcal/mol free energy barrier, which is 9.4 kcal/mol above the free energy of **125c4**. These calculations were not consistent with experimental observations, so alternative intermediates were investigated.

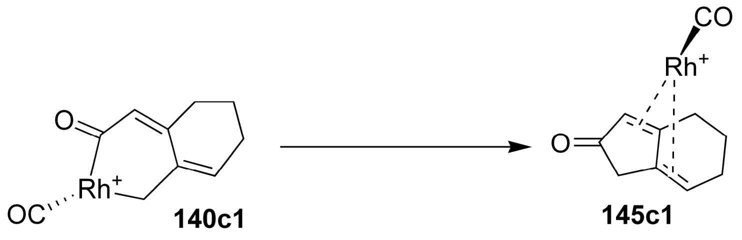


Figure 79 Reductive elimination of rhodium from 140c1.

A carbon monoxide molecule could ligate rhodium in **140c1** before the reductive elimination. This two-step process, shown in **Figure 80**, requires overcoming a smaller free energy barrier than the one-step reductive elimination process described above. The attachment of the carbonyl group to **140c1** was found to proceed without a barrier, resulting in the formation of metallocyclic intermediate **146c1**. In this process the free energy decreased by 16.3 kcal/mol. The next step, the reductive elimination of rhodium from **146c1**, required overcoming a 21.9 kcal/mol free energy barrier.

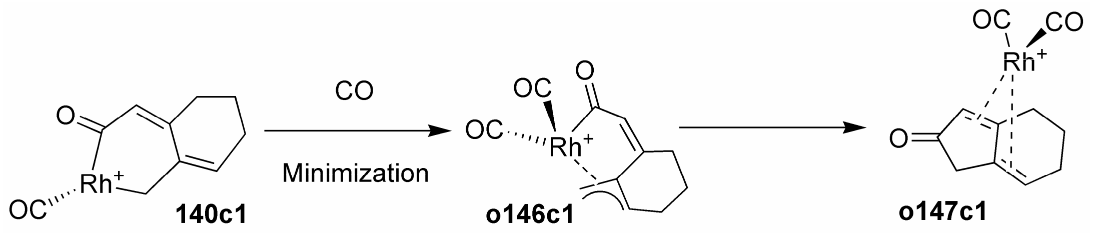


Figure 80 Reductive elimination of rhodium from 140c1 preceded by ligation of rhodium by CO.

Energetics of the Rh-catalyzed cyclocarbonylation pathways described in this appendix are shown in **Figure 81**. The **125c4-127c6-140c1-146c1-147c1** pathway matches the experimental observations. However, the **125c4-127c6-148c1-149c2-147c1** pathway shown in **Figure 58** is the preferred pathway.

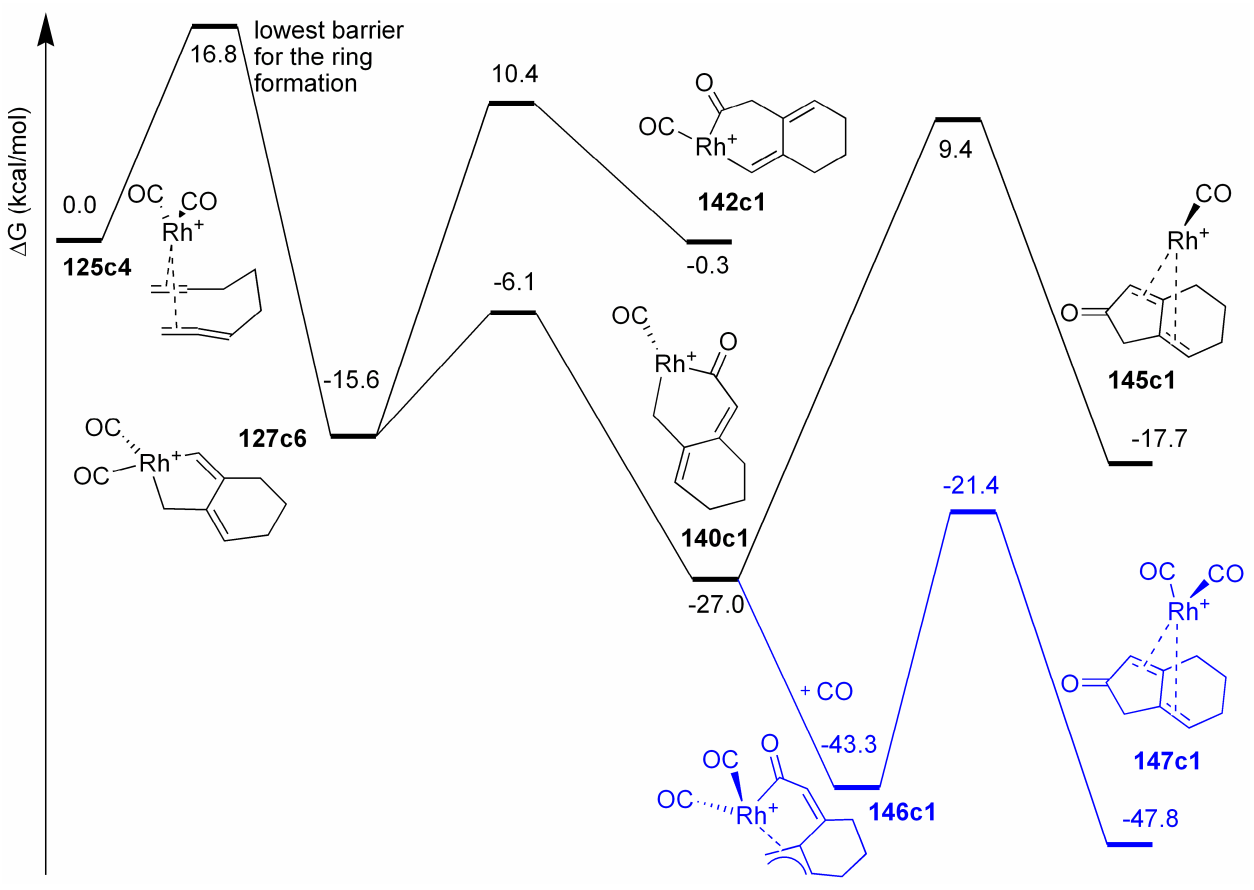


Figure 81 Energetics of the Rh-catalyzed cyclocarbonylation pathways described in this appendix. Parts of pathways with an extra CO present are shown in blue.

Resonance structures are important to key minima in this appendix: **127c6**, **140c1** and **146c1** (**Figure 82**). Based upon the bond lengths listed in **Table 1**, in **127c6** and **140c1** the bond between Rh and C8 has more η^1 character and in **146c1** it has more η^3 character.

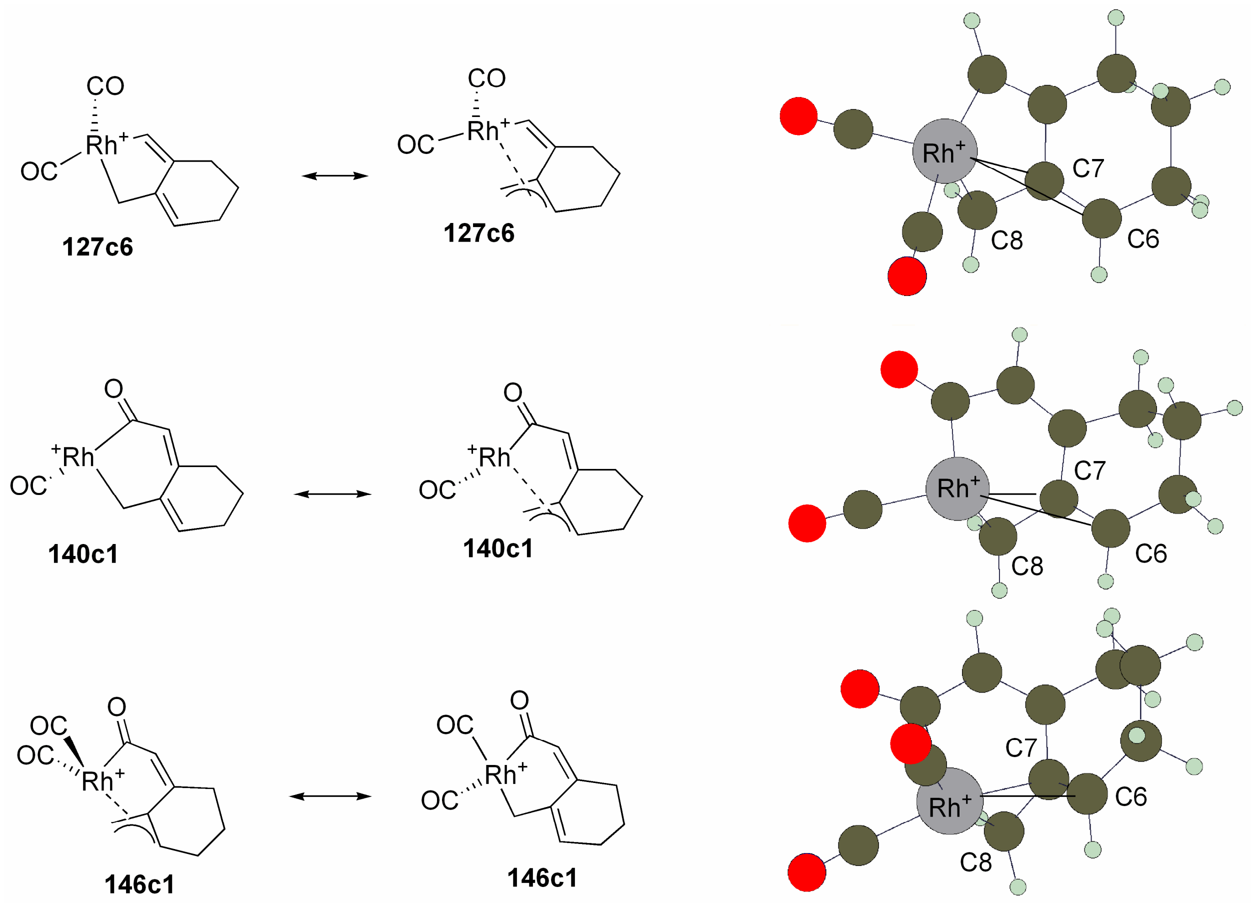


Figure 82 Resonance structures of intermediates 127c6, 140c1 and 146c1.

In 127c6 and 140c1 the bond between Rh and C8 has more η^1 character and in 146c1 it has more η^3 character.

Table 3 Selected bond lengths (in Å) for Intermediates 127c6, 140c1 and 146c1.

Intermediate	C6-Rh	C7-Rh	C8-Rh
127c6	3.462	2.425	2.128
140c1	3.149	2.375	2.039
146c1	2.555	2.220	2.135

APPENDIX D

Oxidative Addition of $Rh^+(CO)_3$

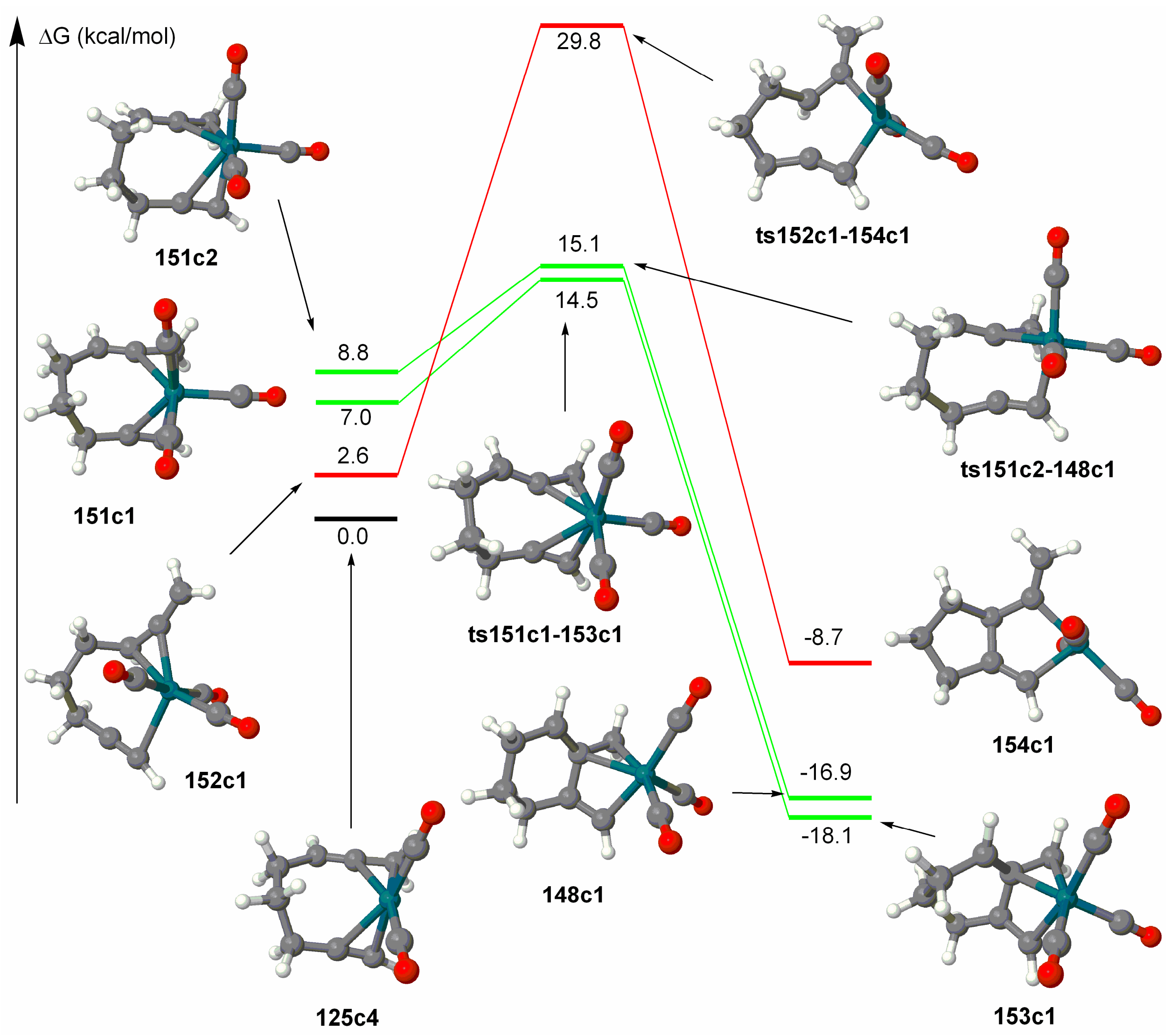


Figure 83 Oxidative addition in the Rh-catalyzed reaction, involving three CO groups.

We considered the possibility that three carbonyl groups ligate rhodium in the oxidative addition step. The molybdenum-promoted reaction had three carbonyl groups involved, so we decided to start with the transition states we found for the molybdenum-catalyzed reaction. We replaced the molybdenum atom with a rhodium cation, fixed the distance between the carbon atoms that were supposed to make a bond and minimized the energy for all other degrees of freedom. We searched for saddle points beginning with the resulting structures. We also searched for transition states using coordinate-driven methods.

One pathway we found led from **152c1** to **154c1**. The transition state for this reaction was 29.8 kcal/mol above **125c4** in free energy, which was somewhat high even for a five-membered ring formation. The other transition state led from **151c2** to **153c1**. Its free energy was only 15.1 kcal/mol above **125c4**. The lowest transition state we found for this reaction was 14.5 kcal/mol above **125c4**. It led from **151c1** to **148c1**.

Most of the three-carbonyl transition states and minima associated with the rhodium-catalyzed ring closing step were closer in structure to the two-carbonyl-containing counterparts than to their molybdenum-containing cousins. Rhodium atoms were not in the plane of the ring system. The exception was the transition state between **152c1** and **154c1**.

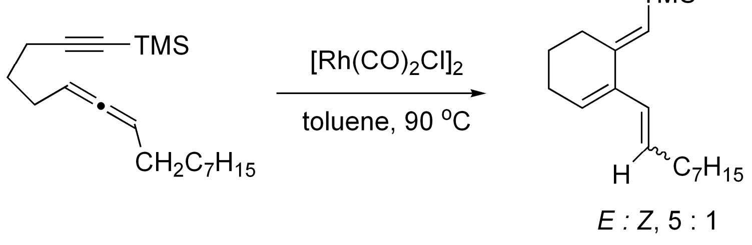


Figure 84 A rhodium-catalyzed cycloisomerization reaction.

Although these transition states were low in energy, we decided to dismiss them for the following reasons:

1) In known catalysis cases, the rhodium atom usually has only one or two carbonyl ligands.

2) If a reaction is done without pumping CO through the solution, the six-membered ring system still forms (**Figure 84**), which indicates that the third CO ligand is not needed.

3) We performed calculations without a solvent. Toluene would have had a more favorable interaction with a system that had only two carbonyl groups.

4) In analogous reactions, bulky steric groups attached to the carbons of the forming ring system speed up the reaction. If the reaction proceeded through a three-carbonyl intermediate, the bulky groups would have slowed it down due to steric hindrance.

That is why on the grounds of experimental evidence we decided that three-carbonyl pathways were not important in the ring forming step of rhodium-catalyzed cyclocarbonylation.

APPENDIX F

A Mechanism for the Rh-Catalyzed Reaction Where CO Inserts into the Allene or Alkyne Prior to Oxidative Addition

The two-CO pathways

A mechanism involving CO insertion into the allene or alkyne, prior to oxidative addition, was considered. The carbons of carbonyl groups of **125c4** were inserted between either C1, C7 or C8 and Rh. Only carbonyl groups *cis* to the carbon to which they were about to bond were used. The insertion of the carbonyl group between the rhodium and C7 was dismissed due to a very high electronic energy barrier of about 37 kcal/mol. Insertion of the carbonyl group between C8 and Rh gave a somewhat unexpected result. In addition to the carbonyl insertion, a bond was formed between C7 and C2, leading to the formation of metallocyclic intermediate **142c3**, which contained a six-membered ring. The free energy barrier for this reaction was found to be 23.6 kcal/mol.

When we started with **125c4** and tried to move the carbonyl carbon towards the terminal carbon in the alkyne group, intermediate **125c4** rearranged into intermediate **160c1**, forming a four-membered ring that includes the carbonyl carbon, the allene and the rhodium. Intermediate **160c1** is a high-energy structure. We found it to be 17.7 kcal/mol higher in free energy than **125c4**. The free energy barrier for the conversion from **125c4** to **160c1** was also high, 30.4 kcal/mol.

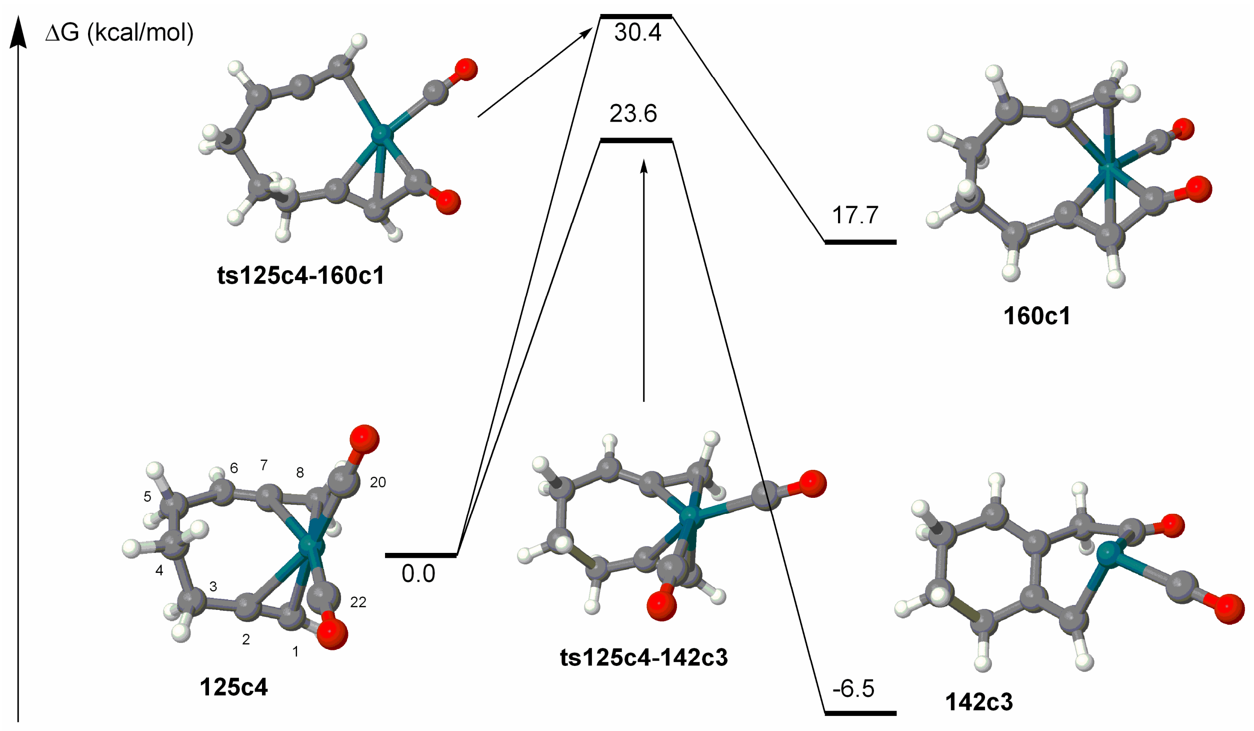


Figure 85 Energetics for transferring a CO out of two from the Rh center to the hydrocarbon chain.

Overall, we found that transferring a CO from the rhodium on **125c4** requires overcoming greater free energy barriers than the oxidative addition converting **125c4** to **127c6**. That is why we decided not to study the remainder of the pathways that begin with the conversion of **125c4** to **142c2** and **160c1**.

The three-CO pathways

Structures in which rhodium is ligated by two or three carbonyls tend to be lower in energy than those in which rhodium is ligated by only one carbonyl. We tried to transfer one carbonyl, leaving the rhodium center with one other carbonyl group. If rhodium could be left with two carbonyls, the products and the transition states for this carbonyl transfer might be lower in energy.

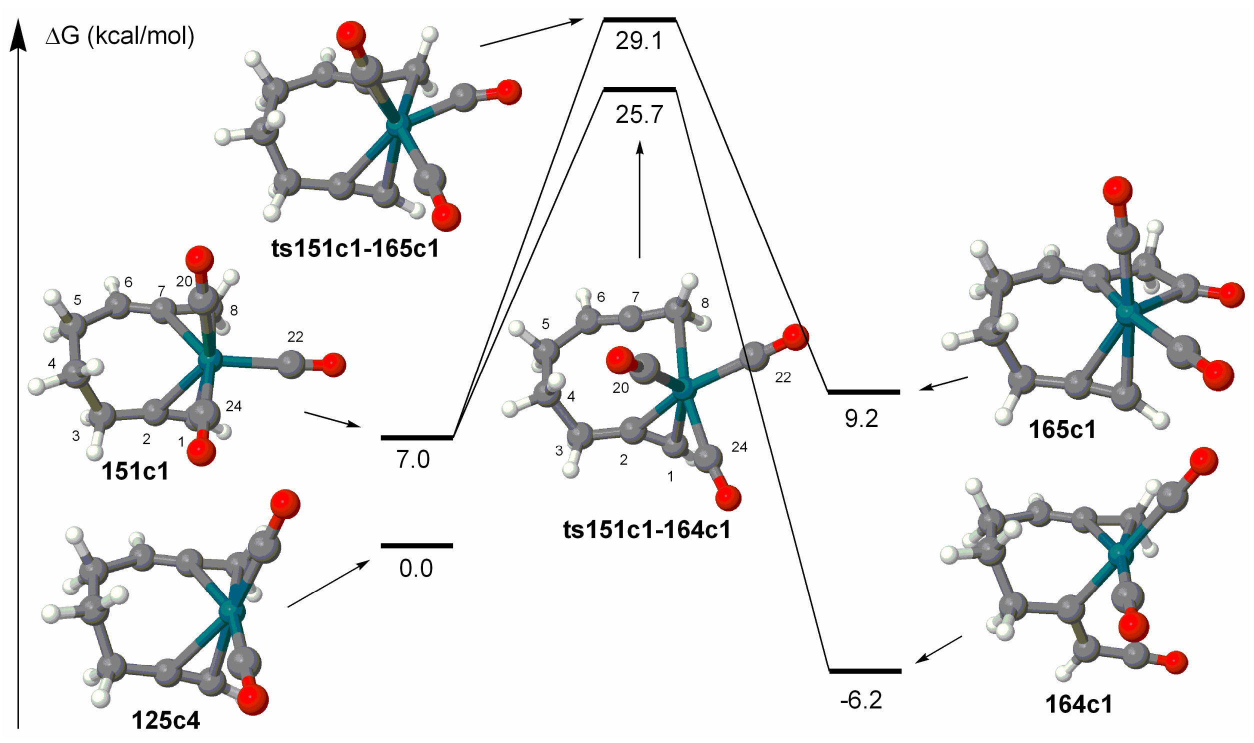


Figure 86 Energetics for transferring one CO out of three from the Rh center to the hydrocarbon chain.

We decided to test this hypothesis. To end up with two carbonyls on rhodium after transferring one carbonyl, we started with **151c1** (**Figure 86**), which is very similar to **125c4** except that it has an extra carbonyl ligand on rhodium. We started with compound **151c1** and looked for transition states and intermediates formed in this reaction step using coordinate-driven methods. The transfer of the CO group to the middle allene carbon was not modeled because the barrier was too high for the analogous reaction beginning with **125c4**. As in the case of the reaction with two carbonyl groups, we considered only the CO groups that were *cis* to the site where they were about to transfer; however, for each such site there were two such carbonyl groups, not just one, as with **125c4**.

When we tried inserting the CO group that was *trans* to the empty site between the alkyl group and the rhodium, we got an unexpected result. The bond between the rhodium and the carbonyl group broke completely. The carbonyl attached itself instead to the alkyl group. The nature of the rhodium-alkyl bond also changed from η^2 to η^1. The product of this reaction, intermediate **164c1,** had a surprisingly low free energy. It was 6.2 kcal/mol below **125c4** (13.2 kcal/mol below **151c1**). The free energy barrier for this reaction was 18.7 kcal/mol. When we tried to transfer the other carbonyl group to a position between the rhodium and the alkyl group, we found the energy barrier to be over 22 kcal/mol relative to **151c1**.

Inserting a carbonyl between the rhodium and the allene also required overcoming large energy barriers. The same transition state and product resulted whether we tried to transfer the CO that is *cis* or the one that is *trans* to the empty site. The free energy barrier height was 22.1 kcal/mol relative to **151c1** (29.1 kcal/mol relative to **125c4)**. The product, **165c1**, was similar to **160c1**. Just like **160c1**, intermediate **165c1** contained a four-membered ring that included the carbonyl carbon, the allene and the rhodium. Intermediate **165c1** was also high in free energy, 9.2 kcal/mol relative to **125c4**.

Transferring the carbonyl group from rhodium before oxidative addition requires overcoming higher free energy barriers than closing the metallocyclic ring and then transferring the CO. The system has to overcome a 23.6 kcal/mol free energy barrier to transfer a CO in the first step, while the barrier for the oxidative addition in the first step is only 16.8 kcal/mol. So the ring must close in the oxidative addition step, as expected.

APPENDIX G

Tables of Energies for Stationary Points Discussed in Chapter 4

Table 4 The computed energies of minima.

Minimum	$E_{electronic}$ (au)	ZPE (au)	E (au)	H (au)	G (au)
CO	-113.30691	-113.30188	-113.29900	-113.29785	-113.32598
125c1	-646.73695	-646.56666	-646.54546	-646.54431	-646.62297
125c4	-646.74257	-646.57261	-646.55138	-646.55023	-646.62921
125c6	-646.74475	-646.57448	-646.55335	-646.55220	-646.63050
126c1	-718.31909	-718.14123	-718.11740	-718.11625	-718.20101
127c4	-646.75278	-646.57896	-646.55985	-646.55870	-646.63230
127c5	-646.77145	-646.59759	-646.57849	-646.57734	-646.65197
127c6	-646.77423	-646.60019	-646.58121	-646.58006	-646.65406
129c1	-718.33146	-718.15099	-718.12873	-718.12758	-718.20967
130c1	-646.74426	-646.56549	-646.54220	-646.54105	-646.62428
130c2	-646.74528	-646.56613	-646.54306	-646.54191	-646.62470
131c1	-718.33106	-718.15230	-718.12901	-718.12786	-718.21109
131c2	-718.33382	-718.15467	-718.13160	-718.13045	-718.21324
131c3	-718.33432	-718.15555	-718.13235	-718.13120	-718.21430
132c3	-718.36439	-718.18225	-718.16074	-718.15959	-718.23905
134	-646.74153	-646.56907	-646.54941	-646.54826	-646.62437
140c1	-646.79667	-646.61982	-646.60197	-646.60082	-646.67221
142c1	-646.75073	-646.57588	-646.55742	-646.55627	-646.62965
142c3	-646.76223	-646.58644	-646.56830	-646.56715	-646.63954
145c1	-646.78401	-646.60608	-646.58865	-646.58750	-646.65738
146c1	-760.14521	-759.97037	-759.95190	-759.95075	-760.02413
147c1	-760.16167	-759.97528	-759.95449	-759.95334	-760.03138
148c1	-760.10647	-759.92487	-759.90212	-759.90097	-759.98400
149c2	-760.13302	-759.94967	-759.92794	-759.92679	-760.00759
151c1	-760.05911	-759.88205	-759.85696	-759.85581	-759.94404
155c2	-718.30862	-718.12654	-718.10535	-718.10420	-718.18247
155c3	-718.30183	-718.11988	-718.09852	-718.09737	-718.17642
156c1	-831.64329	-831.45367	-831.42874	-831.42759	-831.51539
157c1	-718.34609	-718.16106	-718.14082	-718.13967	-718.21617
158c1	-718.31823	-718.13529	-718.11420	-718.11305	-718.19142
159c2	-831.66536	-831.47658	-831.45096	-831.44981	-831.54001
159c3	-831.66183	-831.47402	-831.44811	-831.44696	-831.53766
160c1	-646.71859	-646.54607	-646.52637	-646.52522	-646.60098
162c1	-831.68391	-831.49226	-831.46793	-831.46678	-831.55281
163c1	-831.75401	-831.56005	-831.53667	-831.53552	-831.61993
164c1	-760.08523	-759.90483	-759.88152	-759.88037	-759.96510
165c1	-760.06103	-759.88130	-759.85785	-759.85670	-759.94060

$E_{electronic}$, ZPE, E, H and G stand for electronic energy, zero point energy, energy, enthalpy and free energy respectively.

Table 5 The computed energies of transition states.

TS	$E_{electronic}$ (au)	ZPE (au)	E (au)	H (au)	G (au)	ImagF (cm^{-1})
Oxidative addition step Mo						
ts131c3-129c1, ts131c2-129c1	-718.32126	-718.14170	-718.11978	-718.11863	-718.19966	217.2i
ts126c1-129c1	-718.25723	-718.08168	-718.05909	-718.05794	-718.13980	359.9i
ts131c1-132c3	-718.31370	-718.13519	-718.11320	-718.11205	-718.19191	377.2i
ts131c3-132c3	-718.31006	-718.13168	-718.10966	-718.10851	-718.18858	393.5i
ts126c1-132c3	-718.27768	-718.10010	-718.07787	-718.07672	-718.15990	345.3i
Carbonyl insertion and reductive elimination steps Mo						
ts129c1-159c2, ts129c1-159c3	-831.64192	-831.45572	-831.42961	-831.42846	-831.52190	39.3i
ts159c2-162c1	-831.66386	-831.47463	-831.45008	-831.44893	-831.53695	169.8i
ts162c1-163c1	-831.68172	-831.49050	-831.46699	-831.46584	-831.55000	194.6i
ts159c3-159c2	-831.66044	-831.47263	-831.44767	-831.44652	-831.53465	56.2i
ts129c1-155c3	-718.28172	-718.10160	-718.08021	-718.07906	-718.15920	415.3i
ts129c1-155c2	-718.28589	-718.10571	-718.08448	-718.08333	-718.16259	406.5i
ts155c2-158c1	-718.30508	-718.12299	-718.10273	-718.10158	-718.17775	142.0i
ts158c1-157c1	-718.31301	-718.13031	-718.11015	-718.10900	-718.18510	215.1i
Oxidative addition step Rh						
ts125c4-127c6, ts130c2'-127c6	-646.71824	-646.54822	-646.52861	-646.52746	-646.60236	314.8i
ts125c4-127c4	-646.71452	-646.54486	-646.52504	-646.52389	-646.59981	416.5i
ts125c6-127c5	-646.71482	-646.54482	-646.52521	-646.52406	-646.59896	307.5i
ts130c1-127c6	-646.71327	-646.54304	-646.52345	-646.52230	-646.59704	465.0i
ts130c2-127c5	-646.70883	-646.53909	-646.51929	-646.51814	-646.59377	474.8i
ts130c1-134	-646.70862	-646.53882	-646.51906	-646.51791	-646.59368	277.8i
ts125c1-134	-646.70796	-646.53811	-646.51856	-646.51741	-646.59217	281.1i
Carbonyl insertion and reductive elimination steps Mo						
ts148c1-149c2	-760.11170	-759.93039	-759.91116	-759.91009	-759.98248	221.6i
ts149c2-147c1	-760.11639	-759.93283	-759.91214	-759.91099	-759.98927	334.5i
ts127c6-140c1	-646.76036	-646.58675	-646.56879	-646.56764	-646.63897	268.1i
ts140c1-145c1	-646.73662	-646.56191	-646.54438	-646.54323	-646.61418	456.9i
ts127c6-142c1	-646.73227	-646.55968	-646.54133	-646.54018	-646.61265	350.9i
Carbonyl insertion into allene or alkyne prior to carbocyclization Rh						
ts125c4-142c3	-646.70595	-646.53676	-646.51677	-646.51562	-646.59154	385.3i
ts125c4-160c1	-646.69499	-646.52534	-646.50537	-646.50422	-646.58075	389.1i
ts151c1-164c1	-760.03152	-759.85437	-759.83081	-759.82966	-759.91416	389.2i
ts151c1-165c1	-760.02623	-759.84891	-759.82525	-759.82410	-759.90884	404.0i

$E_{electronic}$, ZPE, E, H and G have the same meanings as in Table 4. ImagF stands for imaginary frequency.

BIBLIOGRAPHY

[1] R.G. Parr, W. Yang, *Density-Functional Theory of Atoms and Molecules* (Oxford University Press, New York, 1989).

[2] V. Sahni, "Quantum-mechanical Interpretation of Density Functional Theory" in **182** *Topics in Current Chemistry* edited by R. F. Nalewajski (Springer, New York, 1996).

[3] Kohn, W.; Becke, A. D.; Parr, R. G. *J. Phys. Chem.* **1996**, *100*, 12974.

[4] March, N. H. *Self-Consistent Fields in Atoms* (Persimmon Press, Oxford, 1975).

[5] Hohenberg, P.; Kohn, W. *Phys. Rev. A*, **1964**, *136*, 864.

[6] Kohn, W.; Sham, L. J. *Phys. Rev. A*, **1965**, *140*, 1133.

[7] Becke, A. D. "Density Functional Theories in Quantum Chemistry" in *The Challenge of d and f Electrons* edited by Salahub, D. R. and Zerner, M. C. (ACS, Washington, D.C., 1989).

[8] Johnson, B. J.; Gill, P. M. W.; Pople, J. A. *J. Chem. Phys.* **1993**, *98*, 5613.

[9] Levine, I. N. *Physical Chemistry 4th Edition* (McGraw-Hill, New York, 1995), p. 663.

[10] Szabo, A; Ostlund, N. S. *Modern Quantum Chemistry, 1st Edition revised* (McGraw Hill, New York, 1982).

[11] Vosko, S. H.; Wilk, L. *Can. J. Phys.* **1980**, *58*, 1200.

[12] Vosko, S. H.; Wilk, L. . *Phys. B: At. Mol. Phys.* **1983**, *16*, 3687.

[13] Becke, A. D. *Phys. Rev. A*, **1988**, *38*, 3098.

[14] Becke, A. D. *J. Chem. Phys*, **1992**, *96*, 2155.

[15] Becke, A. D. *J. Chem. Phys*, **1992**, *97*, 9173.

[16] Lee, C.; Yang, W.; Parr, R. J. *Phys. Rev. B*, **1988**, *37*, 785.

[17] Becke, A. D. *J. Chem. Phys*, **1993**, *98*, 5648.

[18] Perdew, J. P.; Wang, Y. *Phys. Rev. B*, **1992**, *45*, 13244.

[19] Perdew, J. P.; Chevary, J. A.; Vosko, S. H.; Jackson, K. A.; Pederton, M. R.; Singh, D. J.; Fiolhais, C. *Phys. Rev. B*, **1992**, *45*, 6671.

[20] (a) Becke, A. D. *Phys. Rev. A* **1988**, *38*, 3098. (b) Lee, C.; Yang, W.; Parr, R. G. *Phys. Rev. B* **1988**, *37*, 785.

[21] Klamt, A.; Schüürmann, G., *J. Chem. Soc. Perkin Trans.* **1993**, *2*, 799.

[22] Garey, M. R.; Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, (Freeman, San Francisco, 1979).

[23] Stillinger, F. H. Phys. Rev. E, **1999**, 59, 48-51.

[24] Hugh M. Cartwright, *Applications of Artificial Intelligence in Chemistry*, (Oxford University Press, 1993).

[25] Dobson, C. M.; Fersht A. R. *Protein Folding*, Cambridge University Press, 1997.

[26] Deaven, D. M.; Tit, N.; Morris, J. R.; Ho, K. M. *Chem. Phys. Lett.* **1995**, *247*, 339.

[27] Niesse, J. A.; Mayne, H. R.; *J. Chem. Phys.* **1996**, *105*, 4700.

[28] Gregurick, S. K.; Alexander, M. H.; Hartke, B. *J. Chem. Phys.* **1996**, *104*, 2684.

[29] Mesters, J.; Scuseria, G. E.; *J. Comput. Chem.* **1995**, *16*, 729.

[30] http://chemdiv-www.nrl.navy.mil/6110/6112/chemometrics/practga.html

[31] http://guthulamurali.freeservers.com/ga.html

[32] http://panda.ece.utk.edu/~hqi/ece471-571/lecture13_stochastic.pdf

[33] Bäck, T. Evolutionary Algorithms in Theory and Practice: Evolutionary Strategies, Evolutionary Programming, Genetic Algorithms. (Oxford University Press, Oxford, 1996).

[34] Metropolis, N.; Rosenbluth, A. W.; Rosenbluth, M. N.; Teller, A. H.; Teller, E. *J. Chem. Phys.* **1953**, *21*, 1087-1091.

[35] Kirkpatrick, S.; Gelatt, C. D.; Vecchi, M. P. Science (Washington, D.C.) **1983**, 220.

[36] C. J. Geyer, in Computing Science and Statistics: *Proceedings of the 23rd Symposium on the Interface* (American Statistical Association, New York, 1991), p. 156.

[37] Geyer, C. J.; Thompson, E. A. *J. Am. Stat. Assoc.* **1995**, *90*, 909.

[38] Falcioni, M.; Deem, M.W. *J. Chem. Phys.* **1999**, *110*, 1754-1766.

[39] Doll, J. D.; Freeman, D. L. IEEE Comput. Sci. Eng. **1994**, *1*, 22.

[40] Finnila, A. B.; Gomez, M. A.; Sebenik C.; Stenson, C; Doll, J. D. *Chem. Phys. Lett.* **1994**, *219*, 343.

[41] Stillinger, F. H.; Stillinger D. K. *J. Chem. Phys.* **1990**, *93*, 6106.

[42] Wales, D. J.; Doye, J. P. *J. Phys. Chem.* A **1997**, *101*, 5111-5116.

[43] Doye, J. P. K.; Miller, M. A.; Wales, D. J. *J Chem. Phys.* **1999**, *111*, 8417-8428.

[44] Tsai, C. J.; Jordan, K. D. *J. Phys. Chem.* **1993**, *97*, 11227-11237.

[45] Purisima, E. O.; Scheraga, H. A. *Proc. Nat. Acad. Sci.* **1986**, *83*, 2782-2786.

[46] Purisima, E. O.; Scheraga, H. A. *J. Mol. Biol.* **1987**, *196*, 697-709.

[47] Faken, D. B.; Voter, A. F.; Freeman, D. L.; Doll, J. D. *J. Phys. Chem. A*, **1999**, *103*, 9521-9526.

[48] Ji, P.; Wan, Y. F. *International Journal of Computer Applications in Technology*, **2001**, *14*,136-144.

[49] Izrailev, S.; Dimitris, A. *J. Chem. Inf. Comput. Sci.* **2001**, *41*, 176-180.

[50] Mills, G.; Jónsson, H. *Phys. Rev. Lett.* **1994**, *72*, 1124.

[51] Jónsson, H.; Mills, G.; Jacobsen, K. W. *Classical and Quantum Dynamics in Condensed Phase Simulations*, edited by Berne, B. J.; Ciccoti, G.; Coker D. F. (World Scientific, Singapore, 1998).

[52] Maragakis, P.; Andreev, S. A.; Brumer, Y.; Reichman, D. R.; Kaxiras, E. *J. Chem. Phys.* **2002**, *117*, 4651-4658.

[53] Press, W. H.; Teukolsky, S. A.; Vetterling, W. T.; Flannery, B. P. *Numerical Recipes in C++ The Art of Scientific Computing*, 2nd ed. (Cambridge University Press, 2002).

[54] Doye, J. P. K.; Miller, M. A.; Wales, D. J. *J. Chem. Phys.* **1999**, *110*, 6896.

[55] Doye, J. P. K.; Wales, D. J.; Berry, R. S. *J. Chem. Phys.* **1995**, *103*, 4234.

[56] Pillardy, J.; Piela, L. *J. Phys. Chem.* **1995**, *99*, 11805.

[57] Tromp, R. M.; Hamers, R. J.; Demuth, J. E. *Phys. Rev. Lett.* **1985**, *55*, 1303.

[58] Nachtigall, P.; Jordan, K. D.; Janda, K. C. *J. Chem. Phys.* **1991**, *95*, 8652.

[59] Neergaard, H.; Yates, J. T. *Chem. Rev.* **1995**, *95*, 1589.

[60] Kondo, Y.; Amakusa, T.; Iwatsuki, M.; Tokumoto, H. *Surf. Sci.* **2000**, *453*, L318.

[61] Franco, N.; Avila, J.; Davila, M. E.; Asensio, M. C.; Woodruff, D. P.; Schaff, O.; Fernández, V.; Schindler, K. M.; Frizsche V.; Bradshaw A. M.. *Phys. Rev Lett.* **1997**, *79*, 673-676.

[62] Matsumoto, M.; Fukutani, K.; Okano, T. *Phys. Rev. Lett.* **2003**, *90*, 106103.

[63] Wolkow, R. A. *Phys. Rev. Lett.* **1992**, *68*, 2636.

[64] Kondo, Y.; Amakusa, T.; Iwatsuki, M.; Tokumoto, H. *Surf. Sci.* **2000**, *453*, L318.

[65] Yokoyma T.; Takayanagi, K. *Phys. Rev. B*, **2000**, *61*, R5078.

[66] T. Mitsui and K. Takayanagi, *Phys. Rev. B*, **2000**, *62*, R16251.

[67] Hata, K.; Sainoo, Y.; Shigekawa, H. *Phys. Rev. Lett.* **2001**, *86*, 3084.

[68] Hata, K.; Yoshida, S.; Shigekawa, H. *Phys. Rev. Lett.* **2002**, *89*, 286104.

[69] Redondo, A.; Goddard, W. A. *J. Vac. Sci. Technol.* **1982**, *21*, 344.

[70] Shoemaker, J.; Burggraf, J. W.; Gordon, M. S. *J. Chem. Phys.* **2000**, *112*, 2994.

[71] Gordon, M. S.; Shoemaker, J. R.; Burggraf, L. W. *J. Chem. Phys.* **2000**, *113*, 9355.

[72] Jung, Y.; Akinaga, Y.; Jordan, K. D.; Gordon, M. S. *Theor. Chem. Acc.* **2003**, *109*, 268.

[73] Bokes, P.; Stich, I.; Mitas, L.; *Chem. Phys. Lett.* **2002**, *362*, 559.

[74] Konecny R.; Doren, D. J. *J. Chem. Phys.* **1997**, *106*, 2426.

[75] Penev, E.; Kratzer, P.; Scheffler, M. *J. Chem. Phys.* **1999**, *110*, 3986.

[76] Healy, S. B.; Filippi, C.; Kratzer, P.; Penev, E.; Scheffler, M. *Phys. Rev. Lett.* **2001**, *87*, 016105.

[77] Jung, Y.; Shao, Y.; Gordon, M. S.; Doren, D. J.; Head-Gordon, M. *J. Chem. Phys.* **2003**, *119*, 10917.

[78] Buechler E. J.; Boland, J. J. *Science*, **2000**, *290*, 506.

[79] Kresse, G.; Hafner, J. *Phys. Rev. B*, **1993**, *47*, 558.

[80] Kresse, G.; Hafner, J. *Phys. Rev. B*, **1994**, *49*, 14251.

[81] Kresse, G.; Furthmüller, J. *Comput. Mat. Sci.* **1996**, *6*, 15.

[82] Kresse, G.; Furthmüller, J. *Phys. Rev. B*, **1996**, *54*, 11169.

[83] Vanderbilt, D. *Phys. Rev. B*, **1990**, *41*, 7892.

[84] Kresse, G.; Hafner, J. *J. Phys.: Condens. Matter*, **1994**, *6*, 8245.

[85] Monkhorst, H. J.; Pack, J. D. *Phys. Rev. B*, **1976**, *13*, 5188.

[86] Hospital Infection Control Practices Advisory Committee. *Recommendations for Preventing the Spread of Vancomycin Resistance*. MMWR 1995; RR12:1-13.

[87] Singh, M. P.; Janso, J. E.; Luckman, S. W.; Brady, S. F.; Clardy, J.; Greenstein, M.; Maiese, W. M. J. *Antibiotics* **2000**, *53*, 256-261.

[88] Brady, S. F.; Singh, M.; Janso, J.; Clardy, J. *J. Am. Chem. Soc*. **2000**, *122*, 2116-2117

[89] Espinoza, R., Guadamuz, A., Perez, D., Chavarria, F. y Masís, A. **1998**. Species Page de **Daphnopsis americana** (Thymelaeaceae), 9/6/1998. Species Home Pages, Area de Conservación Guanacaste, Costa Rica. http://www.acguanacaste.ac.cr.

[90] Brummond, K.M.; Gao, D. *Organic Lett.*, **2003**, *5*, 3491.

[91] Kent, J. L.; Wan, H.; Brummond, K. M. *Tetrahedron Lett.* **1995**, *36*, 2407.

[92] Frisch, M. J.; Trucks, G. W.; Schlegel, H. B.; Scuseria, G. E.; Robb, M. A.; Cheeseman, J. R.; Montgomery, J. A.; Vreven, T.; Kudin, K. N.; Burant, J. C.; Millam, J. M.; Iyengar, S. S.; Tomasi, J.; Barone, V.; Mennucci, B.; Cossi, M.; Scalmani, G.; Rega, N.; Petersson, G. A.; Nakatsuji, H.; Hada, M.; Ehara, M.; Toyota, K.; R. Fukuda; J. Hasegawa; M. Ishida; T. Nakajima; Y. Honda; O. Kitao; H. Nakai; M. Klene; X. Li; J. E. Knox; H. P. Hratchian; J. B. Cross; C. Adamo; J. Jaramillo; R. Gomperts; R. E. Stratmann; O. Yazyev; A. J. Austin; R. Cammi; C. Pomelli; J. W. Ochterski; P. Y. Ayala; K. Morokuma; G. A. Voth; P. Salvador; J. J. Dannenberg; V. G. Zakrzewski; S. Dapprich; A. D. Daniels; M. C. Strain; O. Farkas; D. K. Malick; A. D. Rabuck; K. Raghavachari; J. B. Foresman; J. V. Ortiz; Q. Cui; A. G. Baboul; S. Clifford; J. Cioslowski; B. B. Stefanov; G. Liu; A. Liashenko; P. Piskorz; I. Komaromi; R. L. Martin; D. J. Fox; T. Keith; M. A. Al-Laham; C. Y. Peng; A. Nanayakkara; M. Challacombe; P. M. W. Gill; B. Johnson; W. Chen; M. W. Wong; C. Gonzalez; Pople, J. A. *GAUSSIAN03*; Gaussian, Inc.: Pittsburgh, PA, 2003.

[93] Hehre, W. J.; Ditchfield, R.; Pople, J. A. *J. Chem. Phys.* **1972**, *56*, 2257.

[94] (a) Hay, P. J.; Wadt, W. R. *J. Chem. Phys.* 1985, *82*, 270. (b) Wadt W.R.; Hay, P.J. *J. Chem. Phys.* **1985**, *82*, 284. (c) Hay P.J.; Wadt, W. R. *J. Chem. Phys.* **1985**, *82*, 299.

[95] Peng, C., Ayala, P. Y. and Schlegel, H.B., *J. Comput. Chem*. **1996**, *17*, 49.

[96] Ayala, P. Y.; Schlegel, H.B. *J. Chem. Phys*. **1997**, *107*, 375–382.

[97] Brummond, K. M.; Chen, H.; Fisher, K. D.; Kerekes, A. D.; Rickards, B.; Sill, P. C.; Geib, S. J. *Org. Lett*. **2002**, 4, 1931.

[98] Yu, Z., Wender, P. A.; Houk, K. N. *J. Am. Chem. Soc*. **2004**, *126*, 9154.

[99] Brummond, K. M.; Chen, H.; Sill, P.; You L. *J. Am. Chem. Soc*. **2002**, *124*, 15186.

[100] Brummond, K. M.; Chen, H.; Mitasev, B.; Casarez, A. D. *Org. Lett.* **2004**, *6*, 2161.

[101] Evans, P. A.; Holmes, A. B. *Tetrahedron* **1991**, *47*, 9131.

[102] O'Hagan, D. *Nat. Prod. Rep.* **1997**, *14*, 637.

[103] Frisch, M. J.; Trucks, G. W.; Schlegel, H. B.; Scuseria, G. E.; Robb, M. A.; Cheeseman, J. R.; Zakrzewski, V. G.; Montgomery Jr., J. A.; Stratmann, R. E.; Burant, J. C.; Dapprich, S.; Millam, J. M.; Daniels, A. D.; Kudin, K. N.; Strain, M. C.; Farkas, O.; Tomasi, J.; Barone, V.; Cossi, M.; Cammi, R.; Mennucci, B.; Pomelli, C.; Adamo, C.; Clifford, S.; Ochterski, J.; Petersson, G. A.; Ayala, P. Y.; Cui, Q.; Morokuma, K.; Malick, D. K.; Rabuck, A. D.; Raghavachari, K.; Foresman, J. B.; Cioslowski, J.; Ortiz, J. V.; Baboul, A. G.; Stefanov, B. B.; Liu, G.; Liashenko, A.; Piskorz, P.; Komaromi, I.; Gomperts, R.; Martin, R. L.; Fox, D. J.; Keith, T.; Al-Laham, M. A.; Peng, C. Y.; Nanayakkara, A.; Gonzalez, C.; Challacombe, M.; Gill, P. M. W.; Johnson, B. G.; Chen, W.; Wong, M. W.; Andres, J. L.; Head-Gordon, M.; Replogle, E. S.; Pople, J. A. *GAUSSIAN98*; Gaussian, Inc.: Pittsburgh, PA, 1998.

[104] (a) Schornack, L. G.; Eckert, C. J. *J. Phys. Chem.* **1970**, *74*, 3014. (b) Dean, J. A. *Handbook of Organic Chemistry*; McGraw-Hill: 1987.

[105] Makino, T.; Itoh, K. *J. Org. Chem.* **2004**, *69*, 395.